ON-LINE ALGORITHMS FOR BIN-COVERING PROBLEMS WITH KNOWN ITEM DISTRIBUTIONS

A Dissertation
Presented to
The Academic Faculty

by

Agni Ásgeirsson

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
May 2014

ON-LINE ALGORITHMS FOR BIN-COVERING PROBLEMS WITH KNOWN ITEM DISTRIBUTIONS

Approved by:

Sigrún Andradóttir, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

David Goldsman, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Anton J. Kleywegt
School of Industrial and Systems Engineering
Georgia Institute of Technology

Pinar Keskinocak
School of Industrial and Systems Engineering
Georgia Institute of Technology

Páll Jensson
School of Science and Engineering
Reykjavík University

Date Approved: 24 January 2014

Perseverance, secret of all triumphs.

Victor Hugo

ACKNOWLEDGEMENTS

The gestation period for this thesis was unusually long, as the time between the thesis proposal defense and the thesis defense was more than thirteen years. A large group of people deserve thanks for supporting and encouraging me during this time.

I would first like to thank my thesis advisors, Professors Sigrún Andradóttir and Dave Goldsman. Without Dave's support and encouragement, I would probably have given up. Sigrún's detailed feedback has made the thesis much better. The rest of the thesis committee deserve praise; Professor Anton Kleywegt gave very good guidance on Markov decision processes, Professor Pinar Keskinocak was very encouraging, and last, but not least, Professor Páll Jensson provided the insight that got me started on this journey more than twenty years ago.

My mom, Albína Thordarson, was absolutely tireless in reminding me that I would never be happy with an unfinished thesis. My siblings, Páll Ágúst Ásgeirsson and Áslaug Ásgeirsdóttir, were always very supportive. My love, Hildur Pétursdóttir, deserves praise for her encouragement during the last push to finish. Lára Jóhannsdóttir provided project management for a year or so, and my stepfather Ólafur Sigurðsson cheered me on as well. My dear daughters, Védís and Iðunn, have been an inspiration to finish before they go to university themselves, and their mother, Eyja Brynjarsdóttir, was always supportive. Cousin Höskuldur and his wife Hilda provided working space on their farm, thank you. It was encouraging to know that my uncle Jón Thordarson was always waiting to go to the defense. Jóhannes Pálsson and Hye Young were very hospitable in letting me stay at their flat in Atlanta. Finally, thanks to Geir Gunnlaugsson, Hörður Arnarson, Jón Þór Ólafsson, and Marel Hf, for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis focuses on algorithms solving the *on-line Bin-Covering* problem, when the items are generated from a known, stationary distribution. We introduce the *Prospect Algorithm*. The main idea behind the Prospect Algorithm is to use information on the item distribution to estimate how easy it will be to fill a bin with small overfill as a function of the empty space left in it. This estimate is then used to determine where to place the items, so that all active bins either stay easily fillable, or are finished with small overfill. We test the performance of the algorithm by simulation, and discuss how it can be modified to cope with additional constraints and extended to solve the Bin-Packing problem as well. The Prospect Algorithm is then adapted to achieve perfect packing, yielding a new version, the Prospect+ Algorithm, that is a slight but consistent improvement. Next, a Markov Decision Process formulation is used to obtain an optimal Bin-Covering algorithm to compare with the Prospect Algorithm. Even though the optimal algorithm can only be applied to limited (small) cases, it gives useful insights that lead to another modification of the Prospect Algorithm. We also discuss two relaxations of the on-line constraint, and describe how algorithms that are based on solving the Subset-Sum problem are used to tackle these relaxed problems. Finally, several practical issues encountered when using the Prospect Algorithm in the real-world are analyzed, a computationally efficient way of doing the background calculations needed for the Prospect Algorithm is described, and the three versions of the Prospect Algorithm developed in this thesis are compared.

CHAPTER 1

INTRODUCTION

This thesis originates for a practical problem faced by Marel, an Icelandic manufacturer of food processing equipment. One of Marel's major products is a machine that weighs items as they are traveling over a conveyor belt scale and, based on each item's weight, the machine will place the item into one of several bins (typically 6–16 bins) on its discharge unit. This machine is called a *grader*, and Figure 1.1 shows such a machine.

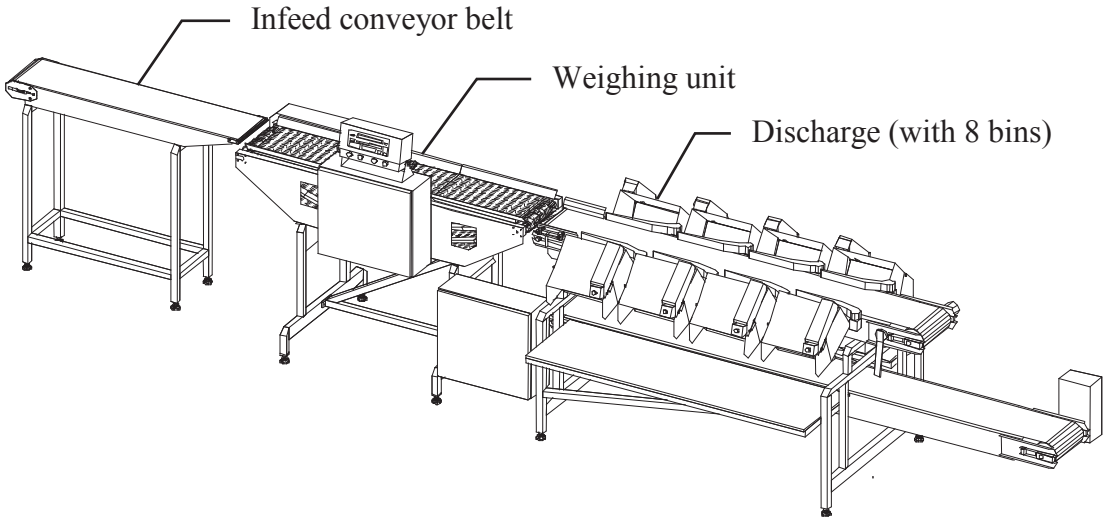


Figure 1.1: Marel grader.

The items can be any product, but usually they come from some natural process where it is impossible to control the weight of the individual items (e.g., chicken pieces, fish fillets, steaks, pork chops, etc.). One of the applications of a grader is to pack the pieces together in packs of guaranteed *minimum* weight. The goal is then to minimize the amount that is given away in excess of the minimum weight, or equivalently, to maximize the number of packs generated. The weight that is

in excess of the minimum weight is called *overfill*. This problem is known in the literature as the *Bin-Covering* problem (sometimes also called the *dual Bin-Packing* problem), and it is closely related to the *Bin-Packing* problem where the goal is to pack the items into packs of a given *maximum* size. These problems are known to be NP-hard (see, e.g., Coffman, Garey, and Johnson [28]). The problem is an *on-line* Bin-Covering problem because the individual pieces must be processed in the order that they arrive, and the machine must decide where to place the newly weighed item before the next item is weighed.

The weight distribution of the items is usually fairly stable in the short term and can be estimated reasonably precisely by the histogram of the last few hundred pieces. It is therefore natural to develop an algorithm that assumes that the item distribution is known and uses this information to assign items to packs. The Prospect Algorithm introduced in Chapter 3 does this, and it has been used with good results in Marel graders. There exist, to our knowledge, no similar algorithms in the Bin-Packing and Bin-Covering literature, and just one journal article written by Litton [90] in 1977 attacks a Bin-Packing problem arising in the carpet industry from a similar angle. It is therefore our opinion that the Prospect Algorithm is a worthy addition to the Bin-Packing and Bin-Covering literature. The main idea behind the Prospect Algorithm is to use information on the item distribution to estimate how easy it will be to fill a bin with small overfill as a function of the empty space left in it. This estimate is then used to determine where to place the items, so that all active bins either stay easily fillable, or are finished with small overfill. The Prospect Algorithm is the main object of this thesis, but as is described in the outline below, other types of Bin-Covering algorithms are discussed, developed, and compared to it. Most comparisons are done with simulations.

review about Bin-Packing and Bin-Covering. In Chapter 3, the Prospect Algorithm is introduced and its performance evaluated with simulations. Chapter 4

explains modifications to the Prospect Algorithm to enable it to solve problems other than the Bin-Covering and Bin-Packing problems covered in Chapter 3. In Chapter 5, we examine how the Prospect Algorithm performs for problems where perfect packing should be possible. This comparison yields a variant on the original Prospect Algorithm, the Prospect+ Algorithm. Chapter 6 describes Markov modeling and Markov Decision Process (MDP) approaches for Bin-Covering. The optimal Bin-Covering strategy is calculated for a problem with a small state space, and from it a new modified Prospect Algorithm is developed and other MDP-related Bin-Covering algorithms are described and tested. In Chapter 7, two different relaxations of the on-line constraint are investigated. These relaxed problems lead to algorithms that are based on solving series of Subset-Sum problems. Chapter 8 describes how the Prospect Algorithm can be modified to work in practical applications and compares the different versions of the Prospect Algorithm in realistic settings. The main thesis then ends with a chapter providing a summary, conclusions, and further research directions. Finally, in the appendices, some detailed simulation results are shown, the random number generator is listed, and simulation data sets are described.

CHAPTER 2

BACKGROUND

This chapter consists of two sections. Section 2.1 states some common definitions from the Bin-Packing and Bin-Covering literature, and Section 2.2 contains a review of that literature. The review is organized as a rough time-line of when new concepts were first introduced in the Bin-Packing/Covering literature.

2.1 Basic Definitions

This section summarizes some common definitions from the Bin-Packing literature (see Coffman et al. [28], and Csirik and Woeginger [40]). The basic problem is defined as follows:

Definition 2.1 (One-Dimensional Bin-Packing Problem) For a given list $L = (a_1, \ldots, a_n)$ of *items* (where a_k denotes the item size and the subscript k is the item index), with $a_k \in (0, 1]$, $\forall k \in \{1, \ldots, n\}$, the goal is to pack the items into a *minimum* number of bins of size 1, such that each bin is at *most* filled to 1.

A related variant is:

Definition 2.2 (One-Dimensional Bin-Covering Problem) For a given list $L = (a_1, \ldots, a_n)$ of *items*, with $a_k \in (0, 1]$, $\forall k \in \{1, \ldots, n\}$, the goal is to pack them into a *maximum* number of bins of size 1, such that each bin is at *least* filled to 1.

The Bin-Covering problem is sometimes also referred to as the *dual Bin-Packing* problem. However, the problem where it is not necessary to pack all the items, and the goal is to maximize the *number* of items used, has also been called *the dual Bin-Packing problem* [58, 101]. The problems are *on-line* if the items arrive in some given

order and each must be assigned to a bin in that order. It is usually assumed that only the size of the next item to be assigned is known (and the current status of the non-empty bins is also known). If the on-line constraint does not apply to a problem, then it is considered to be *off-line*. An on-line problem has *k-bounded-space* if only a fixed number, k, of partially filled bins may be active at any point in the packing process. If this constraint does not apply, then all partially filled bins are active.

Worst-case analysis has been used to evaluate the performance of Bin-Packing/Covering algorithms. For a given list L (as in Definitions 2.1 and 2.2) and algorithm A, let $A(L)$ be the number of bins used when algorithm A is applied to list L and let $OPT(L)$ be the optimal number of bins for a packing of L. Then the *absolute worst-case* performance ratio R_A for algorithm A for the Bin-Packing problem is defined as:

$$R_A = \sup_L \left\{ \frac{A(L)}{OPT(L)} \right\}. \tag{2.1}$$

The corresponding ratio T_A for the Bin-Covering problem is defined as:

$$T_A = \inf_L \left\{ \frac{A(L)}{OPT(L)} \right\}.$$

The *asymptotic worst-case* performance ratio R_A^∞ for the Bin-Packing problem is defined as:

$$R_A^\infty = \limsup_{m \to \infty} R_A^m,$$

where

$$R_A^m = \sup_L \left\{ \frac{A(L)}{OPT(L)} \Big| OPT(L) = m \right\}.$$

The corresponding ratio T_A^∞ for the Bin-Covering problem is defined as:

$$T_A^\infty = \liminf_{m \to \infty} T_A^m,$$

where

$$T_A^m = \inf_L \left\{ \frac{A(L)}{OPT(L)} \Big| OPT(L) = m \right\}.$$

Sometimes there is an upper bound α $(0 < \alpha \leq 1)$ on the size of the items in list L. This leads to the *asymptotic parametric worst-case ratio*, which is defined for the Bin-Packing problem as:

$$R_A^\infty(\alpha) = \limsup_{m\to\infty} R_A^m(\alpha),$$

where

$$R_A^m(\alpha) = \sup_L \left\{ \frac{A(L)}{OPT(L)} \middle| OPT(L) = m \text{ and } a_k \leq \alpha \text{ for all } a_k \in L \right\}.$$

For the Bin-Covering problem, it is defined as:

$$T_A^\infty(\alpha) = \liminf_{m\to\infty} T_A^m(\alpha),$$

where

$$T_A^m(\alpha) = \inf_L \left\{ \frac{A(L)}{OPT(L)} \middle| OPT(L) = m \text{ and } a_k \leq \alpha \text{ for all } a_k \in L \right\}.$$

Performance of Bin-Packing/covering algorithms has also been measured by *average-case analysis*. It is assumed one has a list L_n of n independent and identically distributed (i.i.d.) sizes generated from some probability distribution F. In this case, most functions of the lists L_n become random variables: $OPT(L_n)$, $A(L_n)$, $s(L_n)$ (the sum of the item sizes in L_n), and the performance ratios for Bin-Packing and Bin-Covering are defined the same way, or $R_A(L_n) = T_A(L_n) \equiv A(L_n)/OPT(L_n)$. The average amount of wasted space, $W_A(L_n) \equiv A(L_n) - s(L_n)$ for Bin-Packing and $W_A(L_n) \equiv s(L_n) - A(L_n)$ for Bin-Covering, is also frequently analyzed (recall that the bins are assumed to be of size one, and hence $A(L_n)$ equals the total space in packed bins). For a given algorithm A and distribution F, the following definitions that mirror worst-case analyses are used:

$$\bar{R}_A^n(F) = \bar{T}_A^n(F) \equiv E[R_A(L_n)] = E\left[\frac{A(L_n)}{OPT(L_n)}\right] \tag{2.2}$$

$$\bar{W}_A^n(F) \equiv E[A(L_n) - s(L_n)].$$

The *asymptotic expected ratio* for A under F is defined as:

$$\bar{R}_A^{\infty}(F) = \lim_{n\to\infty} \bar{R}_A^n(F). \tag{2.3}$$

Some of the results listed in the following review are given with $o(g(n))$ (little-o), $O(g(n))$ (big-O), and $\Theta(g(n))$ (big-Θ) notation. Their definitions are:

$f(n) = o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$f(n) = O(g(n))$ if there exist positive constants c and k, such that $|f(n)| \leq cg(n)$ for all $n \geq k$. The values of c and k must be fixed for the function f and must not depend on n.

$f(n) = \Theta(g(n))$ if there exist positive constants c_1, c_2, and k, such that $c_1 g(n) \leq |f(n)| \leq c_2 g(n)$ for all $n \geq k$. The values of c_1, c_2, and k must be fixed for the function f and must not depend on n.

2.2 Literature Review

In this section, the major existing algorithms from the literature for solving Bin-Packing and Bin-Covering problems are described. They often involve scanning the *active bins* in some order (a bin is called *active* if it has at least one piece in it, and the algorithm can still put more pieces into it). A bin is *created* when an item is first put into it. The scanning is usually either done by the order of creation, starting by the oldest bin, or by decreasing order of the content already in the bins, starting with the bin with the largest content (the content of a bin is the total size of all the items already in the bin). The scanning is carried out until either the algorithm has found a place for the item or all active bins have been checked.

The classical Bin-Packing and Bin-Covering problems were categorized as *cutting and packing problems* by Dyckhoff [47] in 1990, along with other related problems: the cutting stock (or trim loss) problem, the strip packing problem, the vehicle, pallet,

or container loading problem, the knapsack problem, etc. In 2005 Wäscher et al. [118] improved on Dyckhoff's classification, and applied their classification to cutting and packing problems from the literature published between 1995 and 2004. To our knowledge, the earliest published work related to cutting and packing is from 1939 by Kantorovich [77] and 1940 by Brooks et al. [18]. Kantorovich discussed Bin-Packing problems arising in organizing and planning production, and Brooks et al. analyzed the 2-dimensional cutting stock problem of cutting rectangles into squares. In 1961 and 1963, Gilmore and Gomory [67, 68] presented a linear formulation for the cutting stock problem, and in 1966, they published another paper on the knapsack problem [69]. Similar algorithms were used by Karmarkar in 1982 [79].

The Bin-Packing problem surfaced in the literature in 1971 as the *loading problem* in an article by Elion and Christophides [49]. They solve a series of small Bin-Packing problems (up to 50 items) with both an exact tree search algorithm from Balas [9] and a heuristic that is similar to the one that later was called the *Best Fit Decreasing (BFD)* algorithm. This is an off-line Bin-Packing algorithm.

In 1972 and 1973, Garay, Graham, and Ullman [65, 66] introduced *worst-case analysis* (see Equation (2.1)) for various heuristic algorithms. The terms *Bin-Packing* and *on-line Bin-Packing* appeared in the early 1970's. The first published article using these terms was written by Johnson in 1974 [76] (he also uses them in his Ph.D. thesis [75]). The author lays the theoretical groundwork for worst-case analysis for the class of heuristic algorithms called *Any Fit* and *Almost-Any Fit*. These classes of algorithms are defined below:

Definition 2.3 (Any Fit (AF) algorithms) An item will not be put into an empty bin unless it does not fit into any of the partially filled bins.

Definition 2.4 (Almost-Any Fit (AAF) algorithms) An item will not be put into the least–filled bin unless it does not fit into any of the other partially filled bins.

Johnson's paper introduces several Bin-Packing algorithms, both on-line and off-line, that have been used in later research. One of them is the *First-Fit (FF)* algorithm. This is an on-line algorithm in which the active bins are indexed by the order of their creation. The bin selection works as follows:

- Starting with the oldest partially filled bin, put the next item in the first bin into which it fits. If the item does not fit into any bin, create a new bin for it.

Best Fit (BF) is a slightly more-advanced on-line algorithm. The partially filled bins are ordered in increasing order by the amount of space left in them. The bin selection works as follows:

- Starting with the partially filled bin with the largest content, put the next item in the first bin into which it fits. If the item does not fit into any bin, create a new bin for it.

Johnson also analyzes the already mentioned BFD algorithm, which is an off-line version of BF that works as follows:

1. Sort the item list in decreasing order.
2. Run the BF algorithm on the resulting item list.

Similarly, *First Fit Decreasing (FFD)* is an off-line version of the FF algorithm. *Next-k Fit (NF_k)* is an on-line bounded-space version of the FF algorithm. There are at most k bins active at any one time. The active bins are indexed by creation order. The bin selection works as follows:

- Starting with the oldest active bin, put the next item in the first bin into which it fits. If the item does not fit into any bin, release the oldest active bin, and create a new bin for the item.

When $k = 1$ this algorithm is usually called *Next-Fit*, or NF, instead of Next-1 Fit or NF_1. NF is basically as simple a Bin-Packing algorithm as one can hope to define since there is no bin decision involved. The algorithm puts the new item in the single active bin if it fits there. Otherwise, it starts a new active bin and puts the item in there instead. The k-bounded-space version of the BF algorithm is *Best-k Fit (BF_k)*. It works as the NF_k algorithm except that the active bins are kept sorted by size, and the BF algorithm is used to find a bin for the item.

Johnson presents the following theorem for any algorithm A that is either Any Fit or Almost-Any Fit:

Theorem 2.5 *[75, 76] For all α satisfying $0 < \alpha \leq 1$,*

1. *If A is an AF algorithm, then $R_{FF}^{\infty}(\alpha) \leq R_{A}^{\infty}(\alpha) \leq R_{NF}^{\infty}(\alpha)$, and*

2. *If A is an AAF algorithm, then $R_{A}^{\infty}(\alpha) = R_{FF}^{\infty}(\alpha)$.*

This means that if one wants to obtain better worst-case results than First Fit, one either needs to relax the on-line constraint or eliminate the AF and AAF constraints. Johnson points out that the worst-case behavior will not yield much information about the average-case behavior. The worst-case bounds given by Johnson have been improved in subsequent papers by Garey et al. in 1976 [64], Liang in 1980 [89], Baker in 1985 [8], Galambos in 1986 [62], Yue in 1991 [120], Vliet in 1992 [116], Mao in 1993 [93], Simchi-Levi in 1994 [112], Dósa in 2007 [46], Xia and Tan in 2010 [119], and Balogh et al. in 2011 [10] . The average-case bounds have been calculated by Shor in 1984 and 1986 [108, 109] for the FF and BF algorithms.

In 1972, Graham [70] and in 1973, Garey, Graham, and Ullman [66] first mentioned the Subset-Sum heuristic, where a Bin-Packing problem is solved by breaking the problem into a series of Subset-Sum problems with the set of currently unpacked items as input. Although the Subset-Sum problem is NP-hard, it is an *easy* NP-hard problem that can be solved optimally with low computational effort for very large

item sets. Vanderbeck [117], Gupta and Ho [45], Flezar and Hindi 2002 [55], Capara and Pferschi [20, 21], and Flezar and Charalambous in 2011 [54] have also put forth or analyzed Subset-Sum heuristics for Bin-Packing.

In 1975 and 1977, Krause, Shen, and Schwetman [84, 85] were the first to analyze Bin-Packing with cardinality constraints when formulating a task-scheduling algorithm on a multiprocessor computer. The cardinality constraint is just on the upper bound of items per bin, i.e., there is a maximum number of items that can be put into a bin. Krause et al. analyzed several approximation algorithms and showed that all had a worst-case performance ratio of 2. In 1999, Kellerer and Pferschy [80] introduced a new heuristic that improved the worst-case bound to 3/2. An on-line version of the problem was explored in 2001 and 2004 by Babel et al. [6, 7], by Epstein in 2006 [50], and by Epstein et al. in 2013 [51]. In 2013, Epstein et al. [51] analyzed Bin-Covering with cardinality constraints, requiring a minimum of $k > 1$ items per filled bin.

In 1977, Litton [90] published a paper on a cutting-stock problem from the carpet industry. A manufacturer holds rolls of carpet in stock, and receives orders from customers for a specific length of carpet, where these orders are less than the length of each roll. The problem is to minimize the scrap (scrap is created when the piece left on a roll is too short to be worth keeping it in stock). Litton models the problem as an on-line Bin-Packing problem and proposes a solution algorithm that uses the average initial roll length and order distribution as inputs. The algorithm also assumes that there is more than one roll of the same carpet to choose from, and it involves setting restrictions on where the carpet rolls can be cut (see Figure 2.1). If a proposed cut is in a no-cut zone, then that order is not cut from the roll and must instead be cut from another roll. The number of no-cut zones is not preset, and to calculate the no-cut zones, Litton models the cutting process with two rolls. If an order is rejected from the first roll, then it is cut from the second roll. The second roll is assumed to

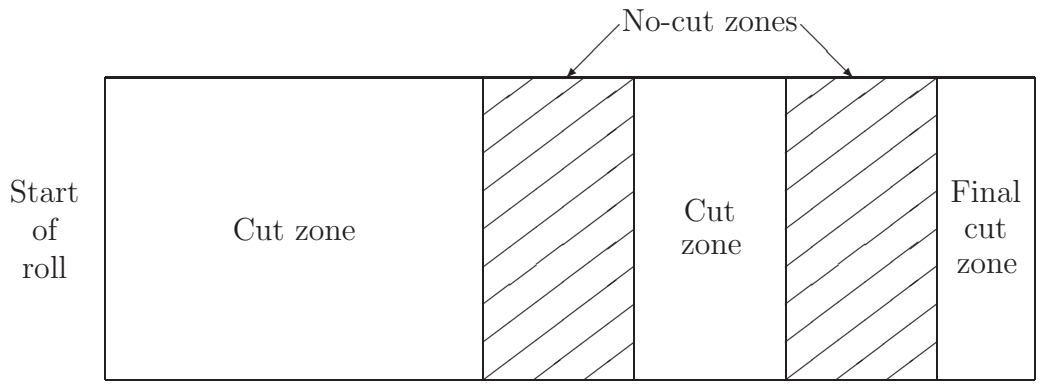


Figure 2.1: Example of Litton's cut and no-cut zones (not to scale).

be infinitely long so that it is guaranteed to accommodate the order. The estimated order distribution is used to calculate the no-cut zones, and the criterion is that if one cuts there, then one will need on average to cut more than the length of a full roll from the second infinite roll before one can cut the first roll into the final cut zone. Litton does not specify the algorithm for this, but instead describes a single example of its input and output. In a real carpet warehouse, where there are no infinite rolls, the cutting process is started on the shortest available roll. If the shortest roll is rejected, then the second shortest roll is checked, and so on. It is not discussed in detail what is to be done if an order is rejected from all rolls, but one can choose to reject the order, put it on hold until the next new roll arrives, or cut it from one of the existing rolls anyway (if a new roll is not arriving soon).

In 1978, Coffman, Garey, and Johnson [26] showed how Bin-Packing methods can be used to solve the minimum *makespan* problem in multiprocessor scheduling, assuming that the tasks are independent and the processors identical (makespan is the total time it takes to run n tasks on m processors, i.e., the time from the start of the first job(s) until the last job is finished). The basic idea is to solve a Bin-Packing problem with bin size C and check if the resulting number of bins is less than or equal to the number of processors m. The goal is then to find the lowest C such that this holds.

In 1980, Coffman et al. [30] modeled the Next-Fit Bin-Packing algorithm as a Markov chain with a continuous state space. This type of modeling is possible because of the simplicity of the NF algorithm. Theoretically, the model works for general item distributions, but practically it is difficult to obtain numerical results except for simple distributions such as the uniform distribution over the interval $(0, 1]$, $\mathrm{U}(0, 1]$. The authors also point out that this type of modeling will quickly become infeasible if the Bin-Packing algorithm is more advanced (and hence more complex). In 1982 Karmarkar [78] improved these types of analysis to include the class of uniform distributions over $(0, a]$, $\mathrm{U}(0, a]$, where $a \leq 1$. That is achieved by converting the differential equation governing the steady-state item distribution into a matrix-differential equation. This allowed him to obtain closed-form expressions for the performance of the Next-Fit algorithm. Karmarkar also introduced the concept of *perfect packing* defined below (see Section 2.1 for the definition of $\bar{W}^n_{OPT}(F)$) and proved that perfect packing can be achieved for any monotonically decreasing density function on $(0, 1]$.

Definition 2.6 (Perfect Packing) A distribution F allows perfect packing if

$$\bar{W}^n_{OPT}(F) = o(n), \ \forall n \geq 1.$$

Note that Definition 2.6 implies that

$$\lim_{n\to\infty} E\left[\frac{OPT(L_n)}{s(L_n)}\right] = 1$$

when the items are drawn independently from the distribution F, since $s(L_n) = O(n)$ (see Coffman, Garey, and Johnson [28]).

In 1982, Knödel [82] introduced the off-line *Pairing* algorithm that works for packing data from a distribution that is symmetric around 1/2. This algorithm sorts the data, and then tries to match together the smallest item with the largest, then the second smallest with the second largest, and so on. If the match is too big for

the bin, the larger item is put in a bin by itself, and the next biggest item is matched with the smallest item. Knödel shows that this algorithm achieves perfect packing for the U$(0, 1]$ distribution. In 1983, Lueker [92] analyzed perfect packing for uniform distributions on $[a, b]$, where $0 < a < b < 1$. He summarized his results as shown in Figure 2.2. The cross-hatched triangle at the top corresponds to $a + b > 1$ and $b > a$ where perfect packing is not possible. The line $a = b$ and the triangle below do not correspond to a meaningful problem. Perfect packing is possible for the line segments $a = 0$ and $a + b = 1$ and on the left boundaries of the shaded diamonds. Perfect packing is not possible inside the shaded diamonds, but the optimum packing ratios in that area can be calculated. The area marked with **?** is not analyzed completely, but Lueker suggests on the basis of numerical experiments that perfect packing is possible there.

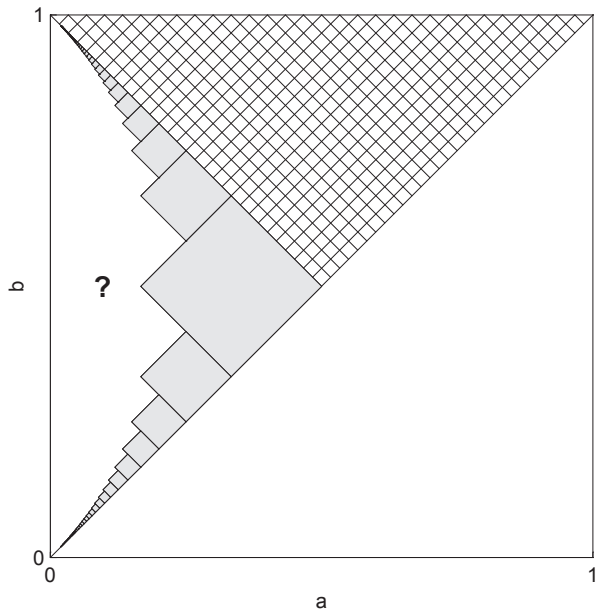


Figure 2.2: Perfect packing for U$[a, b]$ distributions.

Loulou [91] used the same type of algorithm in 1984 to prove perfect packing feasibility of some special non-uniform distributions. The distributions in question are either symmetric around 2^{-p}, for any integer $p \geq 1$, or possess a concave (or

convex) cumulative distribution function. Krause, Larmore, and Volper [83] proved in 1987 that triangular density functions whose expectation is $1/p$ for $p = 3, 4, 5, \ldots$, allow perfect packing as well. The area marked with **?** in Figure 2.2 was proved to allow perfect packing in 1988 by Rhee and Talagrand [100].

In 1984, Bentley et al. [12] showed that the FFD algorithm has bounded expected waste if the items are drawn from the uniform distribution over $[0, \frac{1}{2}]$. The bound they prove is of the order of 10^{10}. In 1991, Floyd and Karp [56] improved this bound to 11.3 by viewing the FFD algorithm as a succession of queueing processes.

A new class of on-line Bin-Packing algorithms was introduced by Hoffmann in 1982 [74] with the *FOLD* algorithm. The idea is to divide the range of the item distribution into sub-intervals and then pair them together strategically so that the items from paired sub-intervals fit well together. The number of sub-intervals, $s \geq 4$, determines the performance of the algorithm, and the size of each sub-interval is based on the item distribution. The larger s is, the better the performance is, but more processing time and computer memory are needed for the algorithm to run. The algorithm is analyzed for two types of distributions, namely, the uniform distribution on $(0, 1]$ and the class of monotonically decreasing density functions on the interval $(0, 1]$. The performance measure that Hoffmann uses is the inverse of the asymptotic expected ratio defined in Equation (2.3), and he proves that the performance of the FOLD algorithm can be made arbitrarily close to the optimum by increasing s.

The *Harmonic*$_M$ and *Refined-Harmonic*$_M$ algorithms were presented by Lee and Lee [87] in 1985. The fundamental difference between the Harmonic$_M$ and FOLD algorithms is that the Harmonic$_M$ algorithm does not pair the M sub-intervals, but assigns one bin to each interval. Each interval can therefore be modeled using the simple Next-Fit Algorithm, and hence worst-case and average-case analyses can be done with relative ease. The Harmonic$_M$ wastes a great deal of space for items larger

than 1/2, and that fault is addressed in the Refined-Harmonic$_M$ by pairing the interval $[1/2, 1]$ with smaller intervals. The authors wrote another unpublished paper [88] in 1987, where they analyzed the Harmonic$_M$ algorithm further and introduced a modified FOLD algorithm called Accelerated-FOLD. This modification is made to improve the asymptotic worst-case performance ratio. Ramanan et al. improved the Harmonic$_M$ algorithm in 1989 [99] with the *Modified Harmonic$_M$* algorithm. This improved the asymptotic worst-case performance. Richey introduced in 1991 [102] another improvement in asymptotic worst-case performance with the Harmonic+1 algorithm. In 2002, Seiden [107] presented a framework for explaining the performance of algorithms based on the Harmonic approach. The author corrected the performance ratio that Richey calculated for the Harmonic+1 algorithm and introduced an improved algorithm called *Harmonic++*.

Bin-Covering was to our knowledge first analyzed in 1984 by Assmann et al. [5]. They call the problem a *dual* version of Bin-Packing. They analyze three algorithms for this problem: *Next-Fit*, *parameterized First-Fit Decreasing* (FFD(r)), and *Iterated Lowest-Fit Decreasing*. The first two algorithms are modified Bin-Packing algorithms. Modification of Next-Fit is trivial — since there is only one active bin, the only thing that changes is when a new bin is created. FFD(r) is a modified Bin-Packing FFD. It is given a parameter $r \in (1, 2)$ and then Bin-Packing FFD is run with r as the (modified) bin size. When all items have been processed, the bins below the real bin size 1 are combined to ensure that all filled bins are above 1 as required. Finally, the basic idea behind Iterated Lowest-Fit Decreasing algorithm is to concentrate on a variant of the following general optimization problem: Given a set I of items and a number of bins N, what is the maximum possible value for the minimum bin level in a packing of I into N bins? Assmann et al. perform both worst-case and average-case analyses of the algorithms for different input distributions.

Csirik and Totik [39] proved in 1988 that all on-line Bin-Covering polynomial-time algorithms have a worst-case performance bound, $T_A^n = 1/2$ (see Section 2.1 for the definition of T_A^n). They also introduced a *perfect covering* theorem similar to the one for Bin-Packing (see Definition 2.6 and Karmarkar [78]) and showed that the uniform distribution on $(0, 1]$ allows perfect covering. In 1991, Csirik et al. [36] performed probabilistic analyses on three different algorithms: NF, *Pairing Heuristic*, and *Next-Fit Decreasing* (NFD). The Pairing Heuristic is modified from Knödel's pairing algorithm for Bin-Packing [82], and the NFD is an off-line version of NF where the items are first sorted by decreasing size before they are run through the NF algorithm. They prove that for the U$(0, 1]$ distribution, NFD has worse average performance than NF.

In 1986 and 1990, Courcoubetis and Weber [31, 32, 33] analyzed the stability of a Bin-Packing system. Their application has item sizes and bin sizes with integer values, and the Bin-Packing system is defined to be *stable* if the total empty space in all unfilled (but not empty) bins is bounded by some number K at all times. Roughly speaking, this amounts to saying that the number of active bins is bounded at all times. They prove a general theorem that covers all discrete distributions F with rational probabilities. Under each such distribution, the optimal expected waste, $\bar{W}_{OPT}^n(F)$, for a random list of n items must be either $\Theta(n)$, $\Theta(n^{1/2})$, or $O(1)$. The task of determining the type of optimal waste is NP-hard. This is basically a discrete version of the perfect packing theorem described on Page 13; the distribution allows perfect packing if the optimal wasted space is either $\Theta(n^{1/2})$ or $O(1)$.

In 1991 and 2000, Coffman et al. [22, 23] noted fundamental discrepancies in the results of average-case analyses of discrete and continuous distributions in Bin-Packing. They analyze the discrete uniform distribution U$\{mj, mk\}$, $1 \leq mj \leq mk$, where each item is from the set $\{1/mk, 2/mk, \ldots, mj/mk\}$ and has probability $1/(mj)$. For fixed j, k, the distribution U$\{mj, mk\}$ informally approaches the continuous distribution

$\mathrm{U}(0, j/k]$ as $m \to \infty$. They show that the results of the average-case analyses for these two related distributions differ substantially. There exist on-line algorithms that have a constant ($O(1)$) wasted space for the discrete distribution, but must have at least $O(\sqrt{n})$ wasted space for the continuous one. FFD has, on the other hand, constant wasted space on $\mathrm{U}(0, u]$, when $u < 1/2$, but there are many combinations of mj, mk with $mj < mk/2$ such that FFD has $O(n)$ expected waste on $\mathrm{U}\{mj, mk\}$.

In 1999, Csirik et al. [38] utilized the optimal expected waste modeling of Courcoubetis and Weber [33] to map the optimal expected waste for the class of *interval distributions* $\mathrm{U}\{h{:}j, k\}, 1 \leq h \leq j \leq k$, in which the item sizes are the integers s, $h \leq s \leq j$, each with equal probability of occurring, and k is the bin size. The results for the $\mathrm{U}\{h{:}j, 100\}$ distribution are summarized in Figure 2.3. Note that this plot is very similar to the uniform distribution version in Figure 2.2. Csirik et al. also introduce a new Bin-Packing heuristic called the *Sum-of-Squares* algorithm (SS), that works on discrete-item distributions where all item sizes and the bin size are integer-valued. Let B be the bin size and let $n(w)$ be the number of active bins whose content totals w, $0 < w < B$ (i.e., these bins have a "gap" of size $B - w$). The selection works as follows:

- Place the next item so as to minimize $\sum_{1 \leq w \leq B} n(w)^2$ for all the active bins.

They compare the waste of this algorithm to the waste of the BF algorithm and the optimal waste for the distribution class $\mathrm{U}\{h : j, 19\}$. Csirik et al. [43] analyzed the SS algorithm further in 2000 and claimed to have proven that with a slight modification to eliminate *dead-end* gap sizes, SS will always achieve the optimal level of wasted space. Dead-end gap size is defined as a gap size that is impossible to fill completely. For example, if the bin size is 6, and the distribution has two item sizes, 2 and 3, then a gap size of 1 is a dead-end. Csirik et al. published the proof in 2006 [44]. In 2001 Csirik, Johnson, and Kenyon [41] extended the SS algorithm to Bin-Covering, and in 2005 they [42] improved the asymptotic worst-case performance ratio bound

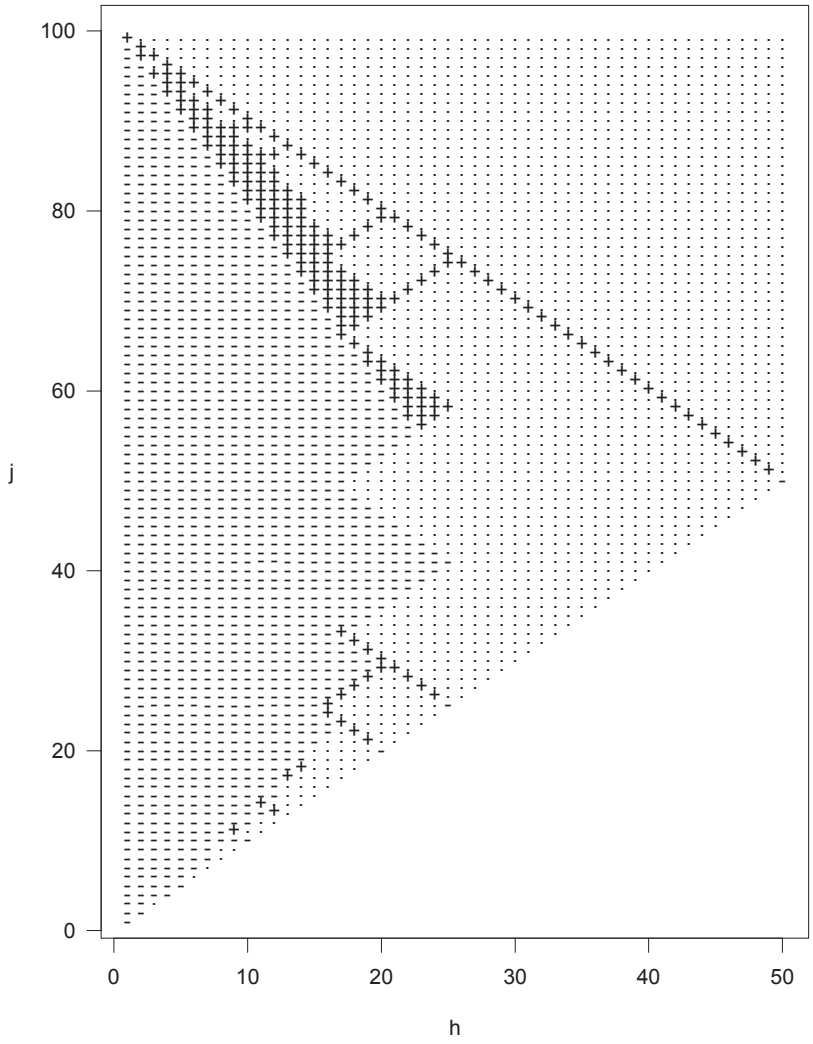


Figure 2.3: Expected optimal waste for all U$\{h{:}j, 100\}$ distributions, where the symbol "-" means $O(1)$, the symbol "+" means $\Theta(\sqrt{n})$, and the symbol "·" means $\Theta(n)$.

of the SS algorithm. Bender et al. in 2007 [11] introduced two faster versions of the SS algorithm, where the slower of the two has the same expected waste space as the original version, but the faster one might have more wasted space. Csirik [34] modeled the on-line Bin-Packing process for the First-Fit algorithm as a random walk in 1986, for the following special item distribution:

$$P(a_i = b) = P(a_i = 1 - b) = 0.5,$$

for some given $b \in (0, 1/2)$. For this item distribution, the expected wasted space is $O(\sqrt{n})$. Csirik and Galambos extended this model in 1986 [37] to cover the uniform $(0, 1]$ distribution by proposing a new algorithm modeled on First-Fit. They called the algorithm *Interval First-Fit*, and it divides the interval $(0, 1]$ into sub-intervals of equal length:

$$I_t = \left(\frac{t-1}{k}, \frac{t}{k} \right], t = 1, 2, \ldots, k.$$

A modified First-Fit is then run on each combined pair of intervals I_t and I_{k-t}, utilizing the fact that the average item size of those combined intervals approaches 1 from below as $k \to \infty$. The expected wasted space of this algorithm is $O(n^{2/3})$.

To our knowledge, the first researchers to analyze Bin-Packing with variable-sized bins were Friesen and Langston [60] in 1986. They present modified versions of the NF and FFD algorithms, and analyze their worst-case bounds. Murgolo [95] attacked this problem in 1987 with a linear programming (LP) based approximation scheme, and Csirik [35] did the same with an on-line algorithm in 1989.

In 1989, Murgolo [96] commented on the anomalous behavior of some Bin-Packing algorithms. It turns out that if there are two lists, L_1 and L_2, where L_2 is derived from L_1 by deleting some elements, some Bin-Packing algorithms can pack L_2 into more bins than they pack L_1. If an algorithm is guaranteed not to do this, it is called *monotonic*, else it is called *nonmonotonic*. Murgolo shows that NF_k is monotonic only for $k \leq 2$, and that BF and BFD are nonmonotonic. Similarly, Graham [70] and

Johnson [75] have shown that FF and FFD are nonmonotonic.

The NF algorithm was improved to some extent by Ramanan [98] in 1989. He proposed the *Smart Next-Fit* algorithm that is identical to NF except when the item to be packed next, a_k, is greater than 1/2 and does not fit into the active bin. In that case, the active bin is closed only if the amount in it is greater than a_k. If not, then a_k is put into an empty bin that is closed immediately. Ramanan compared the average-case performance of NF and SNF for uniform distributions on $[0, b]$, and found SNF to yield a maximum of 7.95% improvement when $b = 1$.

In 1990, Martello and Toth [94] published a book on knapsack problems that has a chapter on Bin-Packing. There they propose an algorithm called *MTP* (Martello Toth Procedure) which is an exact branch-and-bound algorithm accompanied with a couple of preprocessing rules. The initial solution before the branching is obtained by the FFD algorithm. Labbé, Laporte, and Martello [86] proposed a similar algorithm for the Bin-Covering problem in 1995. Scholl, Klein, and Jürgens (1997) designed an exact branch-and-bound algorithm for solving Bin-Packing problems that is more complex than MTP, but with improved performance [105]. In 2006, Fukunaga and Korf [61] proposed bin completion, a branch-and-bound strategy for one-dimensional packing problems, which includes both Bin-Packing and Bin-Covering.

Coffman et al. [24] modeled the Best Fit algorithm as an infinite multi-dimensional Markov chain in 1993. This was done for items from the uniform discrete distribution $\mathrm{U}\{j, k\}$. Albers and Mitzenmacher [1] did similar analyses on the First Fit algorithm in 1998.

In 1995, Falkenauer [52] applied a genetic algorithm to the Bin-Packing problem. Similarly in 1996, Rúnarsson, Jensson, and Jónsson [104] presented a paper on a genetic algorithm for Bin-Covering. In 2013, Sim and Hart [111] published another genetic algorithm for the Bin-Packing problem.

In 1997, Schwering and Wäscher [106] proposed a standardized way of generating

Bin-Packing problems so that future comparisons of Bin-Packing heuristics might become more meaningful. They tested their problem generator with the FFD algorithm and demonstrated how sensitive its performance is to the type of the item distribution. They also criticized Martello and Toth's evaluation of the MTP algorithm [94]. In particular, many of Martello and Toth's test cases were solved by the preprocessing rules and the initial solution algorithm FFD, so Martello and Toth did not really test the performance of the branch-and-bound part of the MTP algorithm.

In 2005, Gutin et al. [72] defined a new variant of the on-line Bin-Packing problem which they called *Batched Bin-Packing*. In this problem, the items arrive in batches and each batch has to be allocated before the next batch arrives. If the batch size is one, this reverts to the classical on-line Bin-Packing problem.

Asgeirsson and Stein in 2006 and 2009 [3, 4] as well as Asgeirsson in 2007 [2] introduced a method to use Markov chains to model a Bin-Covering problem for a given distribution of items. The insights given by the Markov chains is utilized to design a new algorithm, an algorithm that is a heuristic created by simplifying the Markov chain. The Markov chain model they employ is the same as is used in Chapter 6 of this thesis; see Page 80 for further discussion on this.

Several survey articles on Bin-Packing and related problems have been written. To our knowledge, the first was written by Coffman, Garey, and Johnson [27, 28] in 1984 and updated in 1997. A paper on probabilistic analysis of packing and partitioning algorithms was written by Coffman and Lueker [29] in 1991. In 1992, Sweeney and Paternoster [113] wrote a categorized bibliography on cutting and packing problems. Galambos and Woeginger [63] surveyed on-line Bin-Packing algorithms in 1995. In 1997, Dyckhoff, Scheithauer, and Terno [48] published an annotated bibliography on cutting and packing. Csirik and Woeginger [40] wrote a survey of on-line packing and covering problems in 1998. Coffman et al. [25] wrote a survey of bin packing in 1999, its solution algorithms, and some extensions to it.

CHAPTER 3

THE PROSPECT ALGORITHM

3.1 Introduction

In this chapter, the *Prospect* Algorithm is defined, explained, and its performance analyzed. The main idea behind the Prospect Algorithm is to use information on the item distribution to estimate how easy it will be to fill a bin with small overfill, as a function of the empty space left in it. This estimate is then used to determine where to place the items, so that all active bins either stay easily fillable, or are finished with small overfill. Recall that overfill is defined as the amount that is in excess of the minimum weight that is required to be in the bin. First, the Prospect function is introduced in Section 3.2. Then, the Prospect Algorithm is defined in Section 3.3 and its performance is evaluated by conducting simulations for different item distributions in Sections 3.4 and 3.5. Finally, in Sections 3.6 and 3.7, the Prospect Algorithm is adapted to handle Bin-Packing problems as well.

The Bin-Covering problem that is analyzed here is basically the same as the one defined in Definition 2.2, except that it is not normalized to make the bin size equal to one. This is done because it is of interest to analyze the performance of the algorithm as the bin size is varied, but the item distribution is held constant, which would be awkward with the classical definition. It is also assumed that the items are drawn from a known distribution that is stationary over time, and that the problem is subject to the on-line and k-bounded-space constraints (see Section 2.1). Both the item sizes and the bin sizes are assumed to be integers, with 1 as the only common divisor. Note that all problems with rational (or integer) and finite number of item and bin sizes can be converted to this form by multiplication or division with a suitable constant.

3.2 The Prospect Function

The definition of a Prospect function is as follows:

Definition 3.1 A Prospect function for a bin, $\mathrm{Pf}(w)$, is a measure of the probability that the bin eventually will be filled with small overfill as a function of the space w left in it.

Ideally the Prospect function should give the *true* probability that a bin will be filled with small overfill. However, such a function is hard to obtain since it depends not only on the state of the bin but also on the Bin-Covering algorithm being used and the states of the other bins that are available. The focus is therefore on specifying a Prospect function that is simple enough to be easily calculated. Let $\overline{a}$ be the average item size. Intuitively, the Prospect function $\mathrm{Pf}(w)$ should behave as follows:

- For symmetric unimodal item distributions, the closer the ratio $w/\overline{a}$ is to an integer, the better. This is because the ratio $w/\overline{a}$ shows how many average-sized items are needed to fill the bin exactly. For example, it should be better to have room for approximately 2 items left in a bin than to have room for approximately 1.5 items left. This means that $\mathrm{Pf}(w)$ should have local maxima where the ratio $w/\overline{a}$ is integer.

- $\mathrm{Pf}(w)$ should be increasingly sensitive to perturbations of w as w gets smaller. This is because for larger w, there are still many different possibilities for filling the bin, but as w decreases, the number of such possibilities also decreases.

One way to define a Prospect function is to predefine a *zone* just above the bin size; when a bin hits this zone, the bin is *closed* and a new empty bin is made active instead. Then the Prospect function for a single bin can be defined as the probability of reaching the target zone if there is only one active bin, which is the same as saying that the simple Next-Fit (NF) algorithm is used. This proposed Prospect function

has two arguments, i.e., the weight left w and the target zone size z, and is denoted $\mathrm{Pf}(w, z)$. Note that w can be negative, corresponding to a bin that is filled above its capacity. $\mathrm{Pf}(w, 0) = \mathrm{Pf}(w)$ is defined as the probability that a bin of size w will be filled exactly.

$\mathrm{Pf}(w, z)$ can be calculated from $\mathrm{Pf}(w)$ by summing up the probabilities of hitting the different values of the zone. Generally, this cannot be done with a straight sum, $\sum_{i=0}^{z} \mathrm{Pf}(w+i)$, since $\mathrm{Pf}(w+i)$ includes the possibility that the NF process has already hit the zone (the last item has size $\leq i$). An extra term, $-\sum_{j=1}^{i} \mathrm{Pf}(w+i-j)\mathrm{Pf}(j)$, is needed to correct for this:

$$\mathrm{Pf}(w, z) = \begin{cases} 0 & \text{if } w < -z, \\ 1 & \text{if } -z \leq w \leq 0, \\ \displaystyle\sum_{i=0}^{z} \left[\mathrm{Pf}(w+i) - \sum_{j=1}^{i} \mathrm{Pf}(w+i-j)\mathrm{Pf}(j) \right] & \text{if } w > 0. \end{cases} \tag{3.1}$$

Note that if the smallest item in the item distribution, $f_{\min}$, is greater than the zone size z, the extra correction term is always zero, and Equation (3.1) simplifies to (3.2) below. In all simulations done in this thesis, this is the case unless otherwise noted, and hence:

$$\mathrm{Pf}(w, z) = \begin{cases} 0 & \text{if } w < -z, \\ 1 & \text{if } -z \leq w \leq 0, \\ \displaystyle\sum_{i=0}^{z} \mathrm{Pf}(w+i) & \text{if } w > 0. \end{cases} \tag{3.2}$$

The Prospect function $\mathrm{Pf}(w)$ can be calculated using the probability mass function (pmf), $f(w)$, of the item distribution. The function $f^{(c)}(w)$ is defined as the pmf of the item distribution convoluted with itself c times (i.e., it is the pmf of the sum of the sizes of c i.i.d. items), and then $\mathrm{Pf}(w)$ can be calculated using:

$$\mathrm{Pf}(w) = \begin{cases} 0 & \text{if } w < 0, \\ 1 & \text{if } w = 0, \\ \sum_{c=1}^{\infty} f^{(c)}(w) & \text{if } w > 0. \end{cases} \tag{3.3}$$

It should be noted that even though the upper limit on the sum in Equation (3.3) is infinity, in practice the sum generally will not go very high. This is because in "realistic" problems, the pmf function $f(w)$ is usually bounded within finite scalars $f_{\min}$ and $f_{\max}$. This means that $f^{(c)}(w)$ is bounded between $cf_{\min}$ and $cf_{\max}$. Therefore, a set $\mathbb{S}$ can be defined for a given empty space w and zone size z as:

$$\mathbb{S} = \{s \in \mathbb{N} \text{ such that } sf_{\min} \leq w + z \text{ and } sf_{\max} \geq w\},$$

where $\mathbb{N}$ is the set of all positive integers. If the first condition is not true, a sum of any s items from $f(w)$ will always be greater than $w + z$, and hence a bin can not simultaneously contain s items and hit the zone. If the second condition is not true, a sum of any s items from $f(w)$ will always be lower than w, and hence the zone cannot be hit using s items. Now the Prospect can be calculated using:

$$\mathrm{Pf}(w) = \begin{cases} 0 & \text{if } w < 0, \\ 1 & \text{if } w = 0, \\ \sum_{c \in \mathbb{S}} f^{(c)}(w) & \text{if } w > 0. \end{cases} \tag{3.4}$$

Also note that the pmf $f^{(c)}(w)$ only needs to be calculated exactly for low values of c, since as c grows, the Central Limit Theorem implies that the pmf $f^{(c)}(w)$ can be approximated with increasing precision using the normal distribution. Let μ and σ be the mean and standard deviation of the item distribution, respectively. Then, for large c, we have:

$$f^{(c)}(w) \approx \begin{cases} \frac{1}{\sqrt{2\pi c}\sigma} e^{-\frac{1}{2}\left(\frac{w - c\mu}{\sqrt{c}\sigma}\right)^2} & \text{if } w \in \mathbb{N} \text{ and } c \in \mathbb{S}, \\ 0 & \text{else.} \end{cases} \tag{3.5}$$

The Prospect function $\mathrm{Pf}(w)$ can also be calculated using *recursive* convolution by the following theorem:

Theorem 3.2 *$\forall w \in \mathbb{N}$, the values of $\mathrm{Pf}(w)$ can be calculated as follows:*

$$\mathrm{Pf}(w) = \begin{cases} 0 & \text{if } w < 0, \\ 1 & \text{if } w = 0, \\ \sum_{i=1}^{w} f(i)\mathrm{Pf}(w-i) & \text{if } w > 0. \end{cases} \tag{3.6}$$

Proof The cases for $w < 0$ and $w = 0$ are trivial, but the result for the $w > 0$ case needs to be shown. The probability of filling a bin of size $w = 1$ exactly is $\mathrm{Pf}(1) = f(1)\mathrm{Pf}(0)$ since filling it with a single item of size 1 is the only possibility. Similarly, for $w > 1$, $\mathrm{Pf}(w)$ can be calculated as:

$$\begin{aligned} &\mathrm{Pf}(w) \\ &= \sum_{i=1}^{w} \mathrm{P}(\text{the first item is size of } i)\mathrm{P}(\text{a bin of size } w-i \text{ will be filled exactly}) \\ &= \sum_{i=1}^{w} f(i)\mathrm{Pf}(w-i). \end{aligned}$$

■

If the pmf of the item distribution is bounded between $f_{\min}$ and $f_{\max}$, Equation (3.6) can be written as:

$$\mathrm{Pf}(w) = \begin{cases} 0 & \text{if } w < 0, \\ 1 & \text{if } w = 0, \\ \sum_{i=f_{\min}}^{\min(w,f_{\max})} f(i)\mathrm{Pf}(w-i) & \text{if } w > 0. \end{cases} \tag{3.7}$$

By examining Equation (3.7), it is clear that the number of multiplications needed to calculate $\mathrm{Pf}(w)$ is at most $(f_{\max} - f_{\min} + 1)w$. The first term is the maximum number of multiplications of the sum in Equation (3.7), and the second term w is the number of entries of $\mathrm{Pf}(w)$ that are calculated.

Figure 3.1 shows the pmf's of two different discrete distributions, namely, a discrete distribution with rational probabilities that mimics a normal (Gaussian) distribution with $(\mu, \sigma, f_{\min}, f_{\max}) = (100, 15, 55, 145)$, and a discrete uniform distribution $\mathrm{U}_D(75, 125)$. Figure 3.2 shows the corresponding Prospect functions, $\mathrm{Pf}(w)$, calculated using Equation (3.7). Note that the two Prospect functions demonstrate the desired behavior; i.e., peaks occur on even multiples of the mean, and the oscillation fades as w increases.

The above Prospect function for a single bin is the *true* one for the Next-Fit Algorithm, since in this case there are no other bins in the system and the algorithm therefore has no choices on where it can put each item. This means that the Prospect function can be used to calculate the expected average performance of the NF algorithm for the Bin-Covering problem. Each term of the sum in Equation (3.1) with $w = b$ gives the percentage of bins that are filled with weight $b + i$, where b is the bin size and i is the amount of overfill. Therefore, the weight distribution of bins filled via the NF algorithm for Bin-Covering (NFC) with bin size b from items with pmf $f(w)$ is given by the following equation:

$$\mathrm{pmf}_{NFC}(b, i) = \begin{cases} \left[\begin{array}{l} \mathrm{Pf}(b+i) \\ -\sum_{j=1}^{i} \mathrm{Pf}(b+i-j)\mathrm{Pf}(j) \end{array} \right] & \text{if } 0 \le i < f_{\max}, \\ 0 & \text{else.} \end{cases} \tag{3.8}$$

3.3 The Prospect Algorithm

In the preceding section, it was shown how the Prospect function of a single bin can be calculated assuming that this is the only bin. In this section, a Prospect function for a system that has more than one active bin is constructed. Let $\boldsymbol{w} = (w_1, \ldots, w_k)$, where w_i is the empty space in bin i and k is the number of active bins, which is assumed to be constant over time. A decision has to be made about where the next item, a_j, should be placed, and the idea behind the Prospect Algorithm is to do

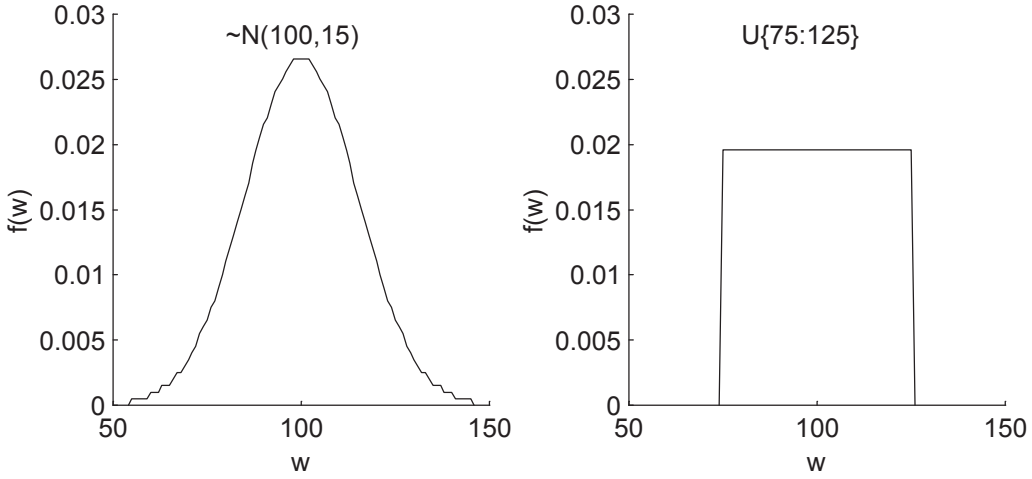


Figure 3.1: Discretized sample distributions.

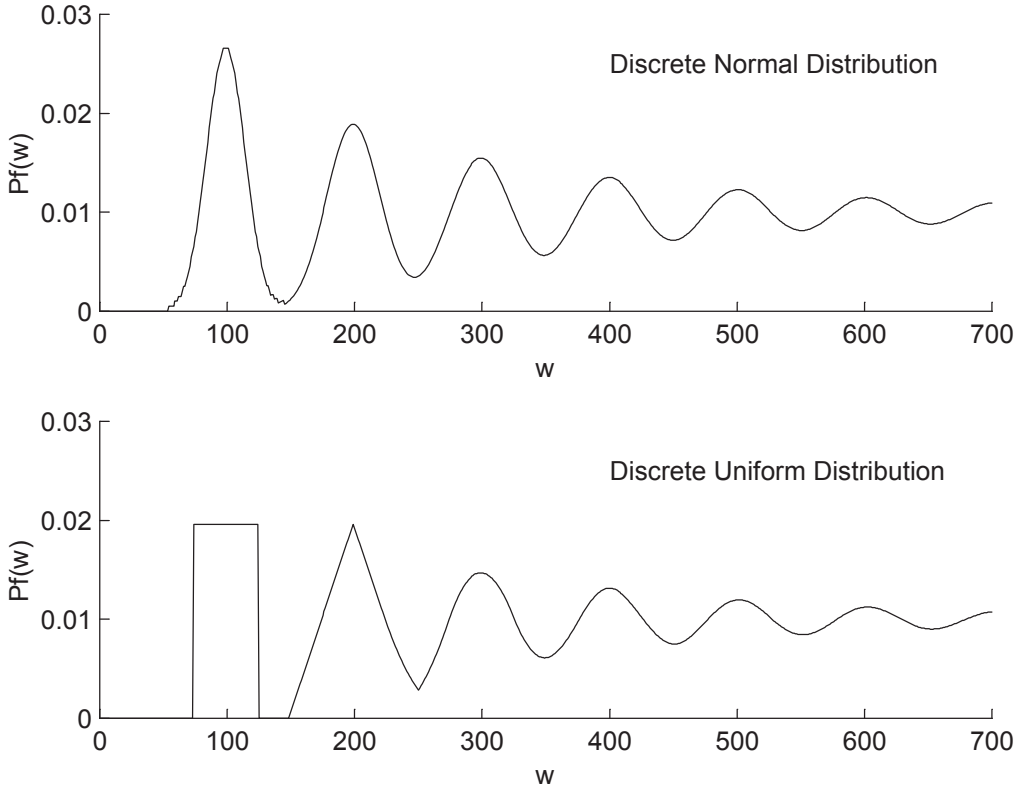


Figure 3.2: Sample prospect functions.

that in such a fashion that a Prospect function for the system of k bins, $\mathrm{Pf}_k(\boldsymbol{w}, z)$, is maximized. Of course, there exist many possible Prospect functions that can be used, but if the cross-correlation between different bins is ignored, then there are two straightforward ways of combining the Prospect functions for single bins, namely, to either multiply or add them together. Multiplication yields the following formulation:

$$\mathrm{Pf}_k(\boldsymbol{w}, z) = \prod_{i=1}^{k} \mathrm{Pf}(w_i, z). \tag{3.9}$$

Now if an item of size a has to be put into one of the k bins, then the bin that will maximize Equation (3.9) has to be found. This amounts to solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \prod_{i=1}^{k} \mathrm{Pf}(w_i - x_i a, z) \\ \text{subject to} \quad & \sum_{i=1}^{k} x_i = 1, \\ & x_i \geq 0 \qquad\qquad (i = 1, 2, \ldots, k), \\ & x_i \text{ is integer} \qquad (i = 1, 2, \ldots, k). \end{aligned} \tag{3.10}$$

If the objective of Equation (3.10) is divided by Equation (3.9) (which is a constant), and it is noted that all terms in Equation (3.10) with $x_i = 0$ cancel out with the corresponding terms in Equation (3.9), then the resulting optimization problem is:

$$\max_{i \in \{1,2,\ldots,k\}} \quad \frac{\mathrm{Pf}(w_i - a, z)}{\mathrm{Pf}(w_i, z)}, \ \forall i, \text{ where } \mathrm{Pf}(w_i, z) \neq 0. \tag{3.11}$$

The expression $\mathrm{Pf}(w_i - a, z)/\mathrm{Pf}(w_i, z)$ is called the Prospect Improvement Ratio (PIR) for bin i. Note that if the prospect of a bin is zero, then that bin has no chance of being finished within the zone, and that is called a *spoiled* bin. PIR is not defined for a spoiled bin.

On the other hand, if the individual bin prospect functions are *summed* together, this yields the following formulation:

$$\mathrm{Pf}_k(\boldsymbol{w}, z) = \sum_{i=1}^{k} \mathrm{Pf}(w_i, z). \tag{3.12}$$

Similar analysis as for the multiplied prospect shows that the resulting optimization problem turns out to be:

$$\max_{i\in\{1,2,\ldots,k\}} \quad \mathrm{Pf}(w_i - a, z) - \mathrm{Pf}(w_i, z). \tag{3.13}$$

The expression $\mathrm{Pf}(w_i - a, z) - \mathrm{Pf}(w_i, z)$ is called the Prospect Improvement Differential (PID) for bin i.

The idea for the Prospect Algorithm is to use one of the optimization problems given in Equations (3.11) and (3.13) as the decision criterion. For the PID model, the resulting Bin-Covering algorithm is:

Algorithm 3.1 (Prospect Differential Algorithm for Bin-Covering)

Step 1: Pick the next item and test-fit it into all active bins. Goto Step 2.

Step 2: Put the item into the bin with the highest PID (ties can be resolved arbitrarily). If the selected bin gets filled, then close it and activate a new empty bin. Go to Step 1.

Note that the above algorithm will not have more than one spoiled bin that is active at any given time, unless the initial empty bins are all spoiled. This is because a bin that is about to be spoiled will have a negative PID; but if there is a bin already spoiled in the system, then it will always have a zero PID, and hence it will be given preference over any bin that is currently not spoiled.

The algorithm for the PIR model is more complicated because a backup decision criterion is needed in the case that the next item will spoil each of the active bins. With the PID model, this is not a problem because the bin with the lowest *original* prospect will have the least negative PID and will therefore be chosen (the original prospect is the current prospect of a bin before the next item is put into it). In the PIR model, all the PIR's will be zero, and arbitrarily resolving those ties is not the best thing to do. The PID is used as the backup decision, or in other words, the bin

with the lowest original prospect is chosen (the worst bin is sacrificed); note that if a bin has already been spoiled, then it will be chosen as with the PID model. For the PIR model, the resulting Bin-Covering algorithm is:

Algorithm 3.2 (Prospect Ratio Algorithm for Bin-Covering)

Step 1: Pick the next item and test-fit it into all active bins. If it has positive PIR for at least one of the bins, then go to Step 2. Otherwise go to Step 3.

Step 2: Put the item into the bin with the highest PIR (ties can be resolved arbitrarily). If the selected bin gets filled, then close it and activate a new empty bin. Go to Step 1.

Step 3: Put the item into the bin with the highest PID (ties can be resolved arbitrarily). If the selected bin gets filled, then close it and activate a new empty bin. Go to Step 1.

In a sense, Algorithms 3.1 and 3.2 will work well if they manage to "surf" the peaks of the Prospect function $\mathrm{Pf}(w, z)$ by keeping unfinished bins close to the peaks in it. The bins will then "surf" on the peaks of the Prospect function until they (hopefully) end up filled within the zone. In the following two sections, the algorithms are tested using simulation.

3.4 Performance Analysis of the Prospect Algorithm

In this section, basic simulation analysis of the Prospect Ratio (PR) Algorithm is undertaken, with the main focus being on the effects of the zone parameter on performance. Corresponding simulations of the Prospect Differential (PD) algorithm show similar results. The simulation method, along with the item distributions used, are described in detail in Appendix A. It is important to keep in mind that the combination of item distribution and bin size is an important factor that determines how low the overfill can become in bin covering. To show that, three discrete approximations of continuous Normal (Gaussian) distributions, labeled $N_D(100,10)$, $N_D(100,15)$, and $N_D(100,20)$, are used. The three distributions are defined in Appendix A, Sections A.2, A.3, and A.4, respectively. The number of bins is set to 8 in all the simulations, the bin size is varied from 200 to 800 in increments of 10, each simulation is run until 20,000 bins have been filled, and the zone parameter is varied from 0 to 100 in increments of 1. For each bin size, the optimal zone parameter that yielded the lowest overfill was selected.

Figure 3.3 shows the resulting overfill plotted as a function of bin size for the three distributions. The overfill curves illustrate the typical pattern of peaks and valleys for low bin capacities. The valleys occur when the bin size is close to an even multiple of the average item size, and the peaks occur when the bin size is between two even multiples of the item average. The peaks are caused by the fact that the distribution does not fit well into the bin size. The peaks become smaller and smaller as the bin size grows, because as the bin size grows, there are more and more different combinations of items that can be used to fill the bins. The difference between the peaks and valleys is inversely related to the standard deviation of the distribution. This is because as the standard deviation grows, the spread of the distribution increases, and there are therefore more combinations of items that can be used to fill the bins.

Setting the zone size, z, is critical for the overfill performance of the Prospect

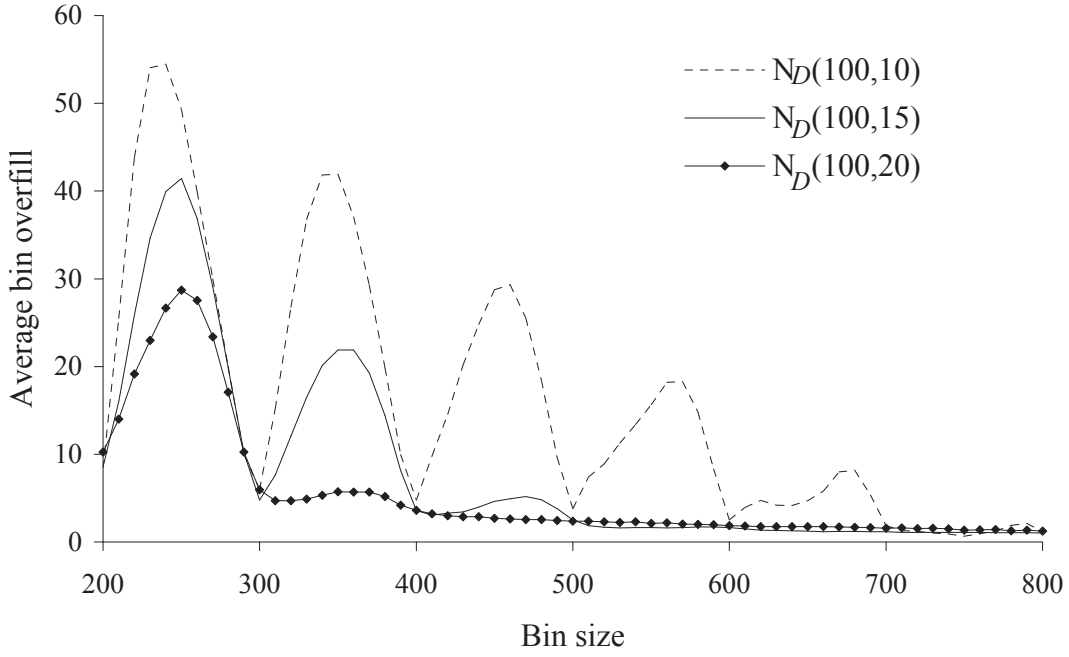


Figure 3.3: PR Algorithm simulations for varying bin sizes and distributions. Eight active bins, and optimal zone value selected for each bin size.

Algorithm. The effects can be classified in detail in two classes: low overfill and high overfill. If the item distribution fits the bin size well, low overfill is possible and the *zonecurve* (overfill as a function of zone size) is shaped as in Figure 3.4. The data for Figure 3.4 are taken from the same simulations as used for Figure 3.3. To better see what is happening, the ratio of spoiled bins (see Page 30) is plotted as well. If the zone size is too low, then the algorithm is forced to spoil too many bins, and those spoiled bins usually have much higher overfill, thus hurting performance. As the zone size increases, overfill from spoiled bins goes down, but on the other hand, overfill from good (or non-spoiled) bins increases. While the spoil rate is high, the effects of decreased spoil rate are more significant than increased overfill of good bins, and therefore, total average overfill goes down. Spoil rate can of course only decrease to zero, so once the spoil rate is close to zero, increasing zone size further will start to increase average overfill. In Figure 3.4, the optimal zone value is 5, where average overfill is 3.56 and spoil rate is 0.637%.

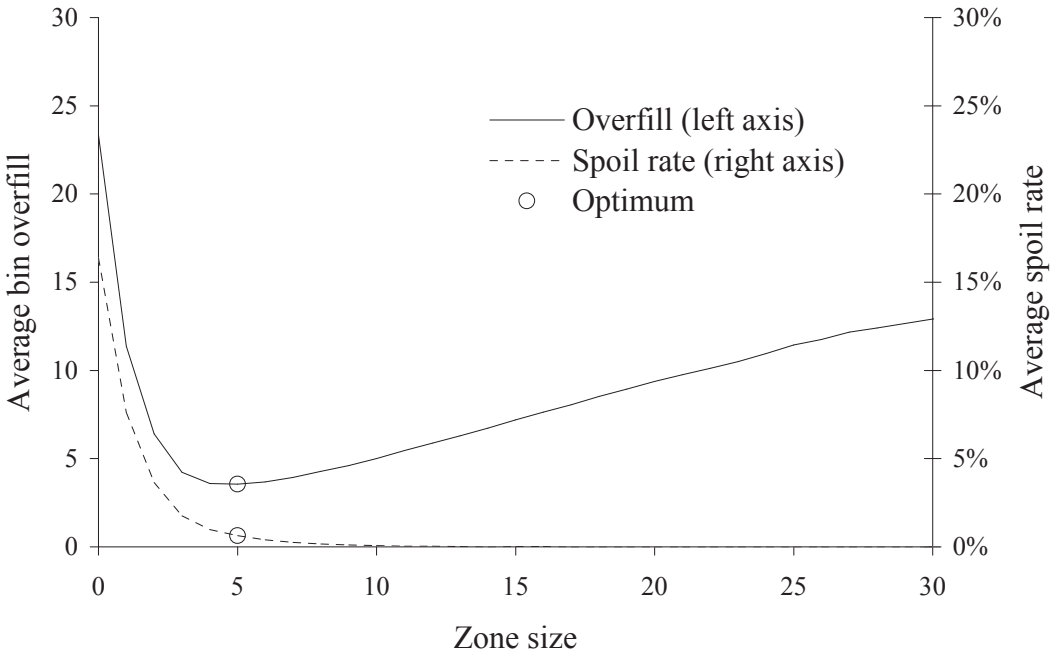


Figure 3.4: Effect of zone size on the overfill and spoil rate of the Prospect Algorithm for eight active bins, bin size 400, and the $N_D(100,15)$ distribution.

Figure 3.5 shows the zonecurve for other bin sizes where low overfill is possible, and all three curves look similar. The solid line in the figure represents the zone size divided by two. Note that as the zone size grows past the optimal value, the average overfill for bin sizes 400 and 600 starts to track the zone size divided by 2, at least for a while. This is typical behavior for low overfill cases where the average number of items per batch is three or more. The reason for this is that the overfill in that case is fairly evenly distributed from 0 to the zone size z, causing average overfill to be around $z/2$.

In the high overfill case, when the item distribution does not fit the bin size well, the zonecurve looks different, and often has two local minimum values as seen in Figures 3.6 and 3.7. The data for the figures are taken from the same simulations as for Figure 3.3. The first local minimum occurs while the spoil rate is still high, but the second minimum occurs when the spoil rate gets close to zero. The second minimum can be explained by the same logic as in the low overfill case before, but the

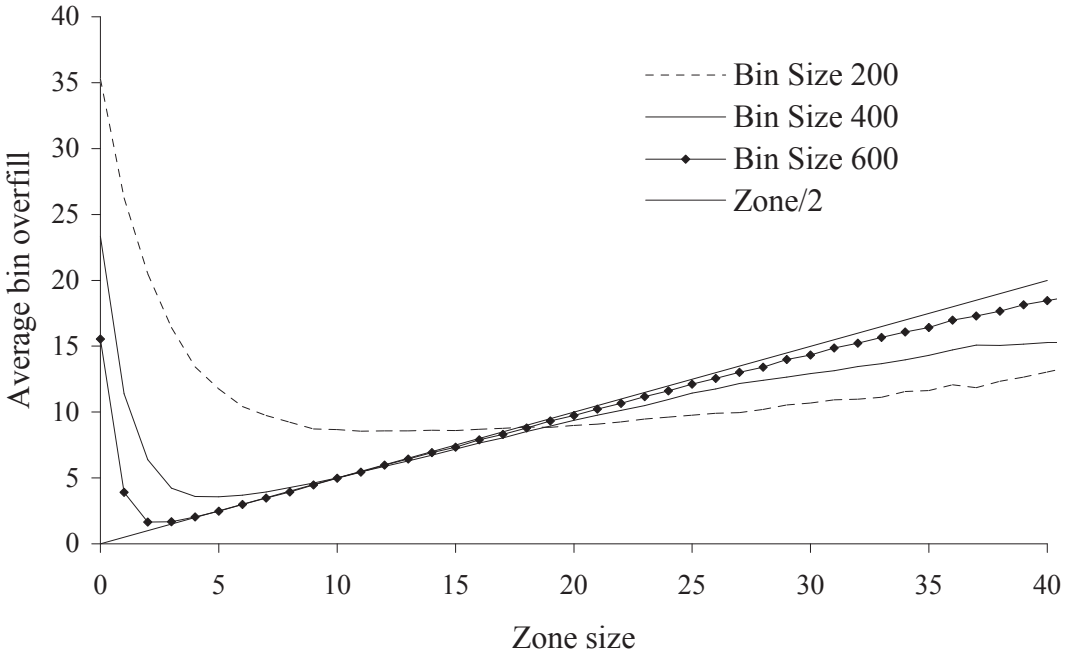


Figure 3.5: Effect of zone size on the PR Algorithm for eight active bins, and the $N_D(100,15)$ distribution.

first minimum can happen because as the zone size increases, the rate of spoil rate decrease is not monotone and can even decrease for a while and then increase again. The first minimum occurs before the rate of spoil rate decrease starts to increase again. This can be observed in the shape of the spoil rate curves, the "hump" located between the two zonecurve local optima. In Figure 3.6, the first (global) minimum occurs when $z = 18$, and the second (local) minimum occurs when $z = 61$. In Figure 3.7, the first (local) minimum occurs when $z = 9$, and the second (global) minimum occurs when $z = 47$.

In Figure 3.3, the optimum zone parameter was determined by running many (101) repeated simulations where only the zone parameter is varied (from 0 to 100). The number of repetitions can be cut down significantly by monitoring if it is highly probable that a local minimum has been found and then stopping the simulations. This can easily be done by monitoring during the repeated simulations the spoil rate and the difference between the current zone size and optimum overfill so far. The

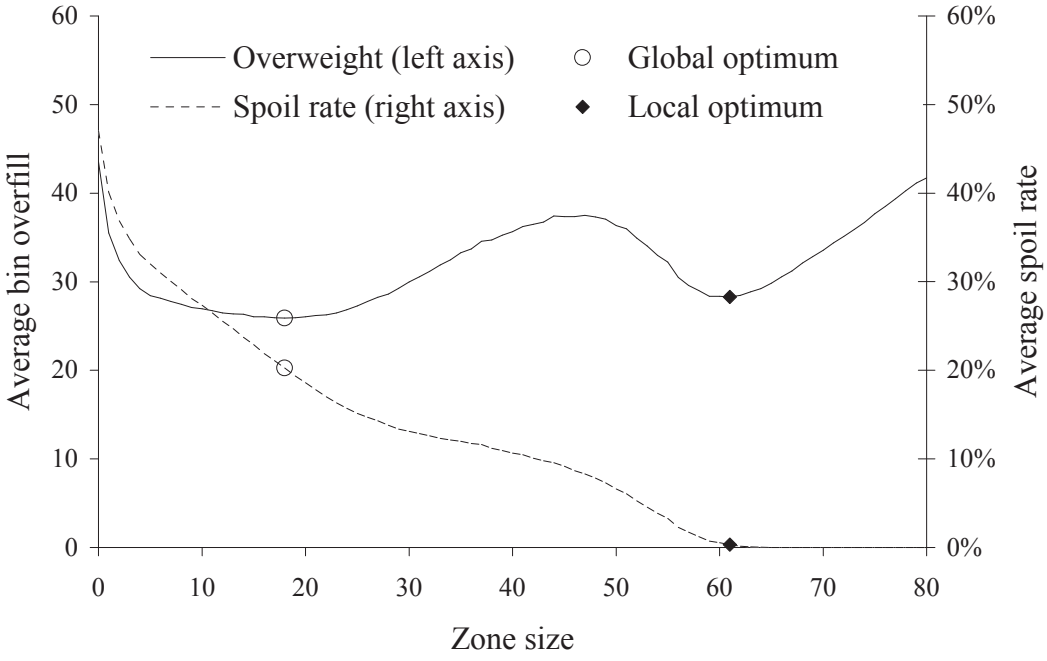


Figure 3.6: Effect of zone size on the PR Algorithm for eight active bins, bin size 220, and the $N_D(100,15)$ distribution.

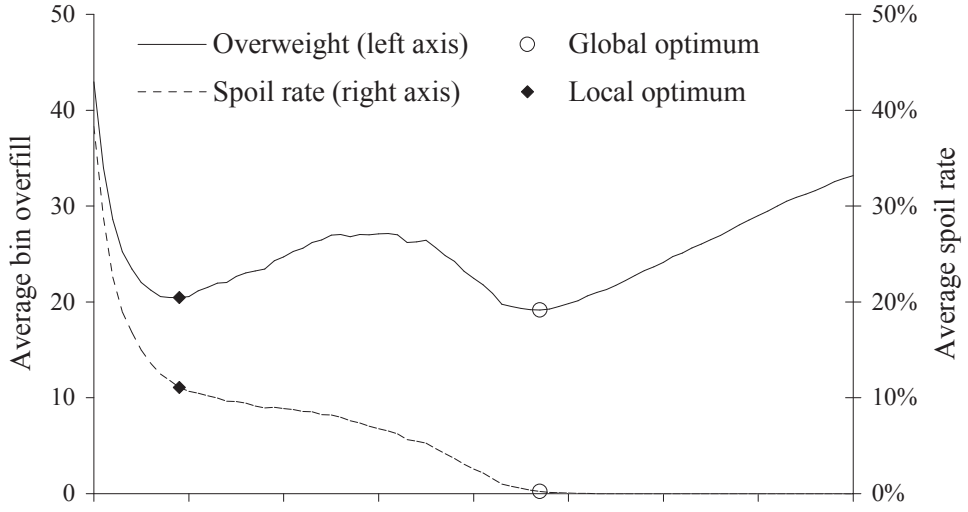


Figure 3.7: Effect of zone size on the PR Algorithm for eight active bins, bin size 220, and the $N_D(100,20)$ distribution.

following algorithm has been developed for this:

Algorithm 3.3 (Optimal Zone Search Algorithm)

Step 1: Start with zone size $z = 0$. Go to Step 2.

Step 2: Run the simulation, and log key results such as average overfill o, spoil rate s, the optimal zone value so far $\hat{z}$, and the corresponding overfill $\hat{o}$. Go to Step 3.

Step 3: Check the stopping criteria; stop if it returns true. Else increment z by 1 and go to Step 2. The stopping criteria is: $(s < 0.01\%)$ and $(\hat{o} < z/2)$ and $(z - \hat{z} > 5)$.

The key is the set of stopping criteria, which consists of three parts. The first check monitors the spoil rate and ensures that it is almost zero; the optimum is highly unlikely to occur after the spoil rate is this close to zero. The second part adds a safety margin that is inspired by how the overfill often starts to track $z/2$ when $z > \hat{z}$, as seen in Figure 3.5. The third part is an extra safety measure ensuring that at least 5 zone increases are run after the most-current optimum value is found. Algorithm 3.3 was tested on the simulation data that was generated for Figure 3.3, and it selected the optimal zone value in all cases.

Finally, Figure 3.8 shows how the number of active bins affects the performance of the PR Algorithm. Simulations are run on the same data as in Figure 3.4, the number of active bins is 4, 8, and 16, the bin size is varied from 200 to 800 in increments of 10, and the optimal zone is determined using the Optimal Zone Search Algorithm 3.3. As expected, the performance improves with added bins.

3.5 Comparison of the Next-Fit, PR, and PD Algorithms

In this section, Next-Fit and the two versions of the Prospect Algorithm are tested and compared. The three item distributions from the previous section are used, i.e.,

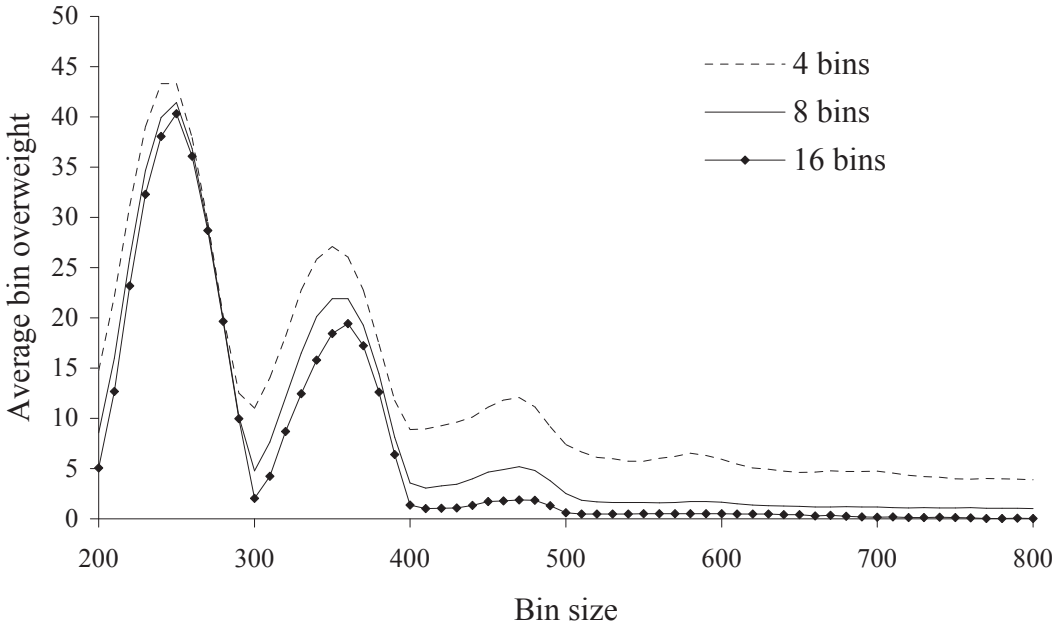


Figure 3.8: Effect of the number of active bins on the PR Algorithm for the $N_D(100,15)$ distribution.

$N_D(100,10)$, $N_D(100,15)$, and $N_D(100,20)$, together with a discrete uniform distribution labeled $U_D(75,125)$ (see Appendix Section A.9). Each simulation starts up with a warmup period lasting until 1,000 bins have been filled, and after that the simulation is run until $20{,}000 * 30 = 600{,}000$ bins have been generated. The bins are grouped into 30 batches of 20,000 bins, and the average overfill is calculated for each batch. This is then used to the calculate the average overfill and standard deviation of the overfill. In Figures 3.9 through 3.12, the average overfill is plotted for Algorithms 3.1 (PD Algorithm), 3.2 (PR Algorithm), and the Next-Fit Algorithm with varying bin size from 200 to 800 in increments of 20. The overfill values for the Next-Fit Algorithm are calculated using Equation (3.8), the other ones are simulation averages. A 95% confidence interval for the average overfill is calculated for each run, but it is not included on the plot, since it is so tight that it is about the same as the thickness of the line. There are eight active bins. The optimal zone size for each simulation is determined with the Optimal Zone Search Algorithm 3.3. The performance gains of

the Prospect Algorithms compared to the Next-Fit Algorithm are obvious.

The performance of the PR and PD Algorithms is close in most cases except for the $N_D(100,10)$ distribution, where the PD Algorithm does significantly worse. Also, for the $U_D(75,125)$ distribution, the PD Algorithm fares consistently better, but the difference is slight. In order to quantify the difference between the two versions, the 95% confidence intervals for the average bin overfill are compared. The results are summarized in Tables 3.1 and 3.2. A detailed comparison for Table 3.1 is listed in Appendix B. The values in Table 3.2 are calculated from the average and standard deviation of the overfill for all the 930 batch means simulations for each distribution. Overall, the PR Algorithm is better 22.6% of the time, PD is better 37.9% of the time, and 39.5% of the time there is a statistical tie. The advantage of PD in this comparison stems from the Uniform distribution, but the PD advantage in average overfill is only 0.03%. The results for the normal distributions favor PR slightly in Table 3.1. In Table 3.2, there is a statistically significant difference in the average overfill in favor of PR for the $N_D(100,10)$ and $N_D(100,20)$ distributions (0.19%, and 0.01%, respectively), but a statistical tie for the $N_D(100,15)$ distribution. Based in the four sets of experiments, PR has lower overfill than PD, with the difference being 0.04%. The conclusion is that the PR Algorithm seems preferable to the PD Algorithm, since it is more stable than PD, and has a slightly lower average overfill in our simulations. In subsequent analysis in the thesis, the PR variant is used, and is called simply the "Prospect Algorithm".

3.6 Prospect Function for Bin-Packing

So far the Prospect Algorithm has only been applied to Bin-Covering problems, but it can be modified for the Bin-Packing problem as well. Note that Equations (3.6) and (3.8) are not correct for Bin-Packing, since the distribution of the first item in an empty bin is not the same as the item distribution of subsequent bins. The larger an

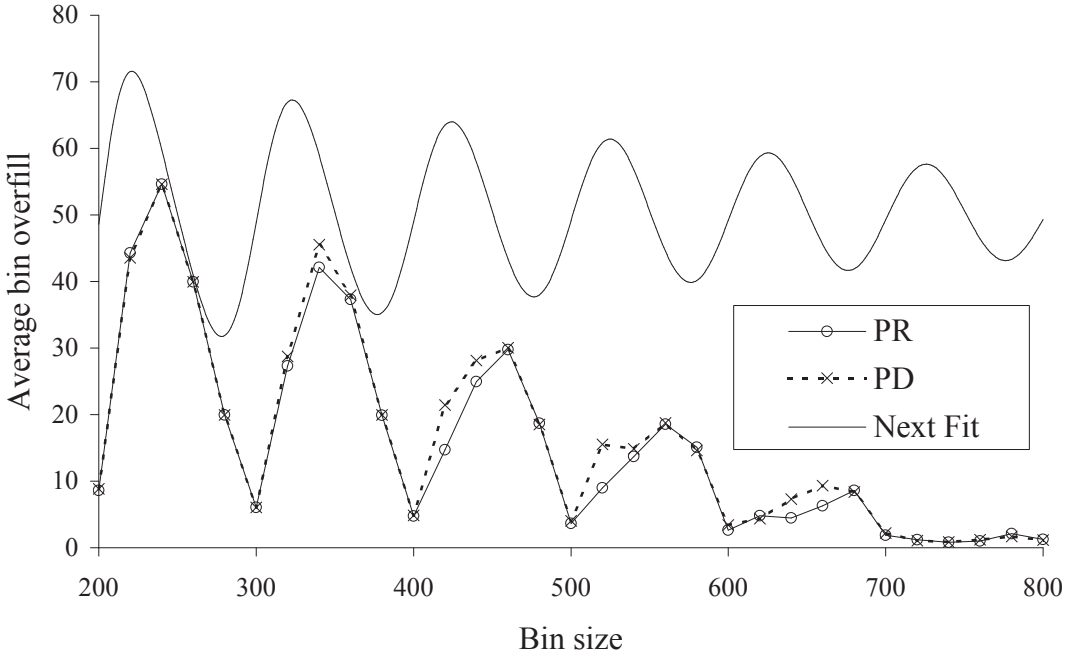


Figure 3.9: Effects of varying bin size for eight active bins and the $N_D(100,10)$ distribution.

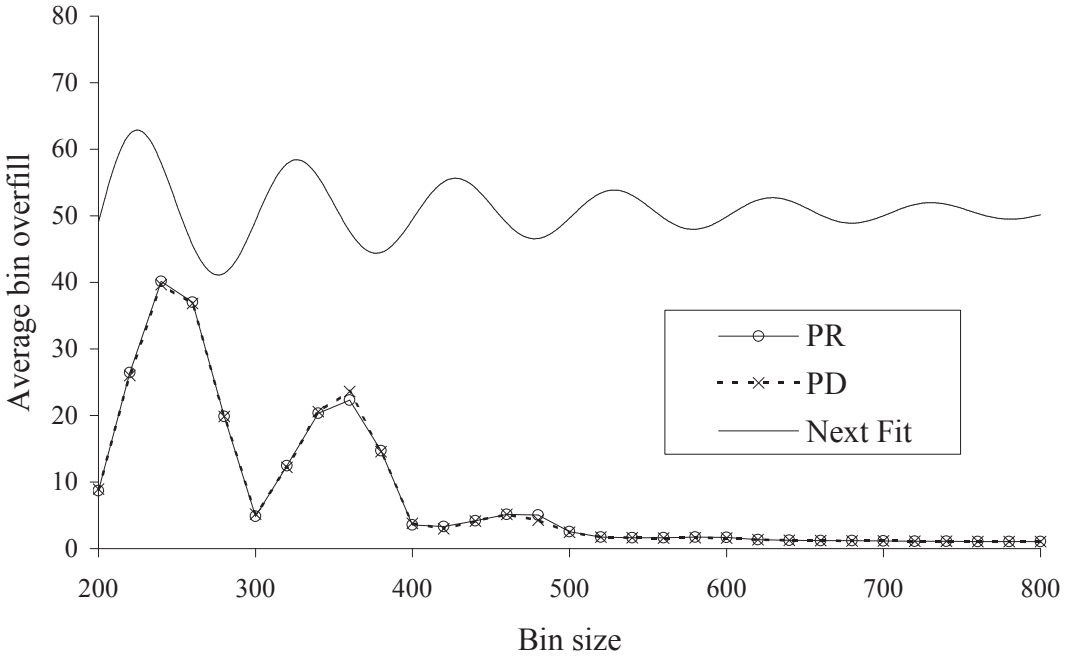


Figure 3.10: Effects of varying bin size for eight active bins and the $N_D(100,15)$ distribution.

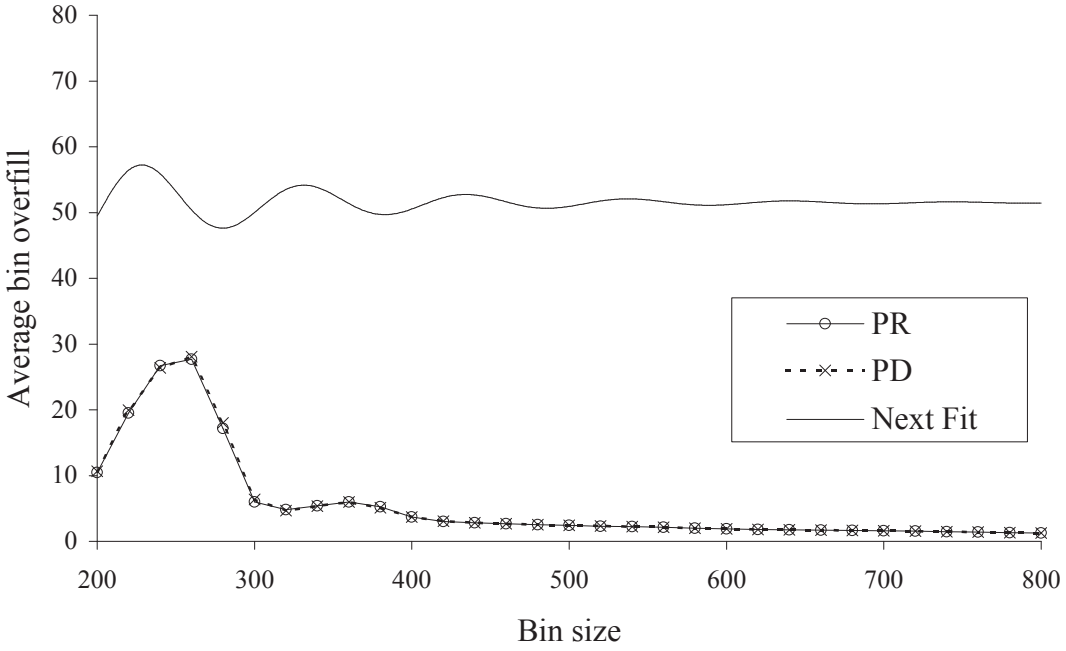


Figure 3.11: Effects of varying bin size for eight active bins and the $N_D(100,20)$ distribution.

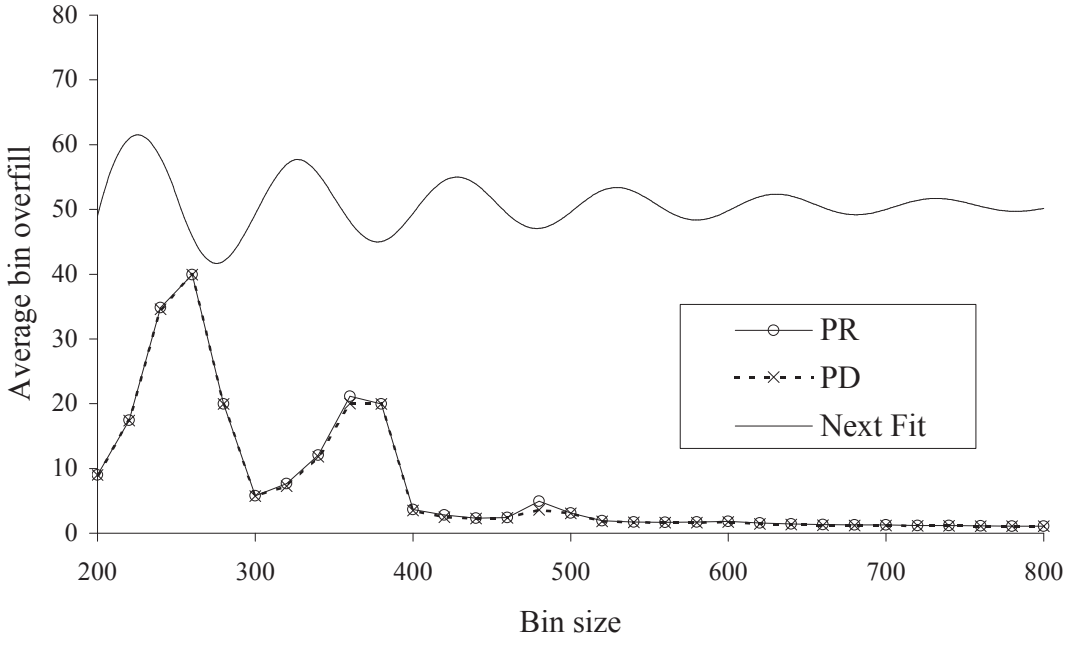


Figure 3.12: Effects of varying bin size for eight active bins and the $U_D(75,125)$ distribution.

Table 3.1: "Statistical winners" summary of the PR and PD Algorithms.

	Winner		
Distribution	PR	TIE	PD
$N_D(100,10)$	18	5	8
$N_D(100,15)$	5	14	12
$N_D(100,20)$	5	23	3
$U_D(75,125)$	0	7	24
Total	28 (22.6%)	49 (39.5%)	47 (37.9%)

Table 3.2: Summary of average overfill for the PR and PD Algorithms (entries are in %).

Distribution	PR	PD
$N_D(100,10)$	$[3.146, 3.151]$	$[3.332, 3.336]$
$N_D(100,15)$	$[1.614, 1.618]$	$[1.611, 1.614]$
$N_D(100,20)$	$[1.110, 1.113]$	$[1.119, 1.122]$
$U_D(75,125)$	$[1.472, 1.475]$	$[1.440, 1.443]$
Total	$[1.836, 1.838]$	$[1.876, 1.878]$

item is, the more likely it is to become too big for the active bin, and therefore become the first item in a new empty bin. This phenomenon is sometimes called the *inspection paradox*, see, e.g., [53]. Calculating the probability mass function of the size of the first item in a bin, $f_1(w)$, is not trivial. However, the following approximation applies for the distribution of the size of the first item in a bin for continuous distributions as the bin size (b) goes to infinity [103]:

$$\lim_{b\to\infty} f_1(w) = \frac{w}{\mu} f(w). \tag{3.14}$$

This formula seems to work for discrete distributions as well. The different first-item distribution causes the Prospect function, $\widehat{\mathrm{Pf}}(w)$, to be different than for the Bin-Covering problem ($\mathrm{Pf}(w)$). Assuming that $\widehat{\mathrm{Pf}}(w)$ is known, then $f_1(w)$ can be calculated by summing up the probabilities that an item of size w does not fit in a

bin:

$$f_1(w) = f(w) \sum_{i=1}^{w-1} \widehat{\mathrm{Pf}}(b-i), \tag{3.15}$$

where b is the bin size, $f(w)$ is the probability of getting an item of size w, and the sum is the probability of the bin having less than w left to be filled. Assuming $f_1(w)$ is known, then $\widehat{\mathrm{Pf}}(w)$ can be calculated by convoluting the distribution with the Bin-Covering Prospect function $\mathrm{Pf}(w)$:

$$\widehat{\mathrm{Pf}}(w) = \sum_{i=f_{\min}}^{\min(w,f_{\max})} \mathrm{Pf}(w-i) f_1(i). \tag{3.16}$$

Now $f_1(w)$ and $\widehat{\mathrm{Pf}}(w)$ can be calculated by iterating between Equations (3.15) and (3.16). In particular, the procedure first estimates $f_1(y)$ using Equation (3.15) with $\widehat{\mathrm{Pf}}(w) = \mathrm{Pf}(w)$, then gets a new estimate of $\widehat{\mathrm{Pf}}(w)$ with Equation (3.16), then gets a new estimate of $f_1(w)$, and so on. If $f_1(w)$ converges, a solution is found. What conditions, if any, needed on the item distribution to guarantee convergence are not known at this point, but to date the procedure has always converged. The iterations are run until the maximum difference between single values of $f_1(w)$ between iterations is less than 10^{-16}. Note that in order to make the iterations converge to $f_1(w)$, each new $f_1(w)$ needs to be normalized to ensure $\sum_{i=f_{\min}}^{f_{\max}} f_1(i) \equiv 1$ since rounding errors can cause the sum to diverge from 1, resulting in loss of convergence.

As an example, $f_1(w)$ is calculated for the distributions in Figure 3.1 and compared to the item distribution $f(w)$ and the approximation from Equation (3.14). The results are show in Figures 3.13 and 3.14. In both cases the bin size is 200, and distribution $\mathrm{N}_D(100,15)$ is used for Figure 3.13 and $\mathrm{U}_D(75,125)$ for Figure 3.14. There is an obvious difference in the three distributions, with the Equation (3.14) estimate lying between $f_1(w)$ and $f(w)$. In Figures 3.15 and 3.16, the bin size is varied from 200 to 800, and the average of $f_1(w)$ is calculated for each run. The average of $f(w)$ and the Equation (3.14) estimate are also plotted. The average for $f_1(w)$ is always

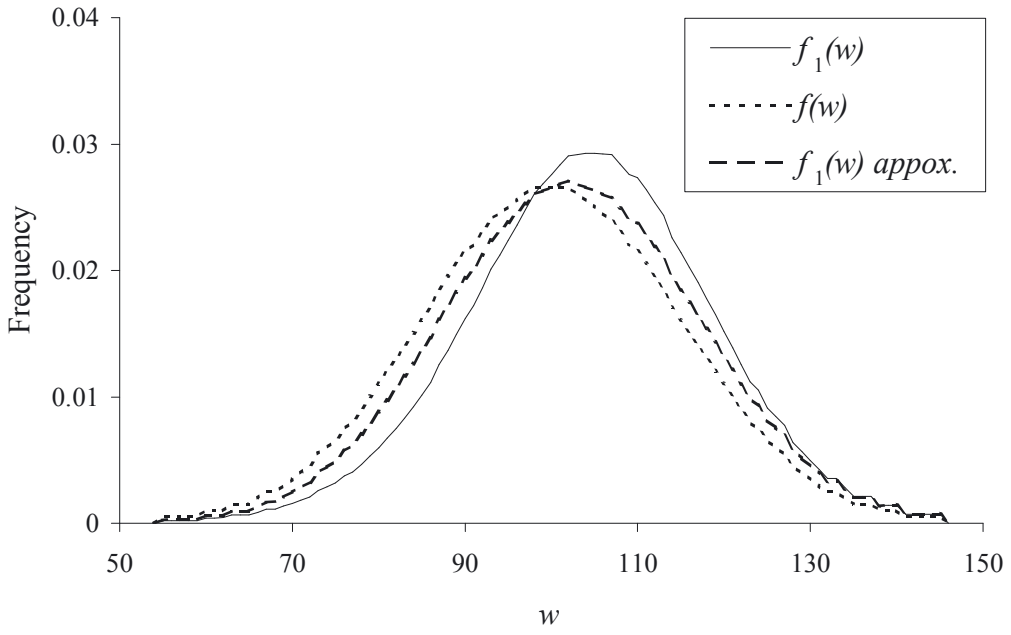


Figure 3.13: Distribution of first item $f_1(w)$ in Bin-Packing with bin size 200 and item distribution $f(w)$ being $\mathrm{N}_D(100,15)$. The Equation (3.14) approximation is included as well.

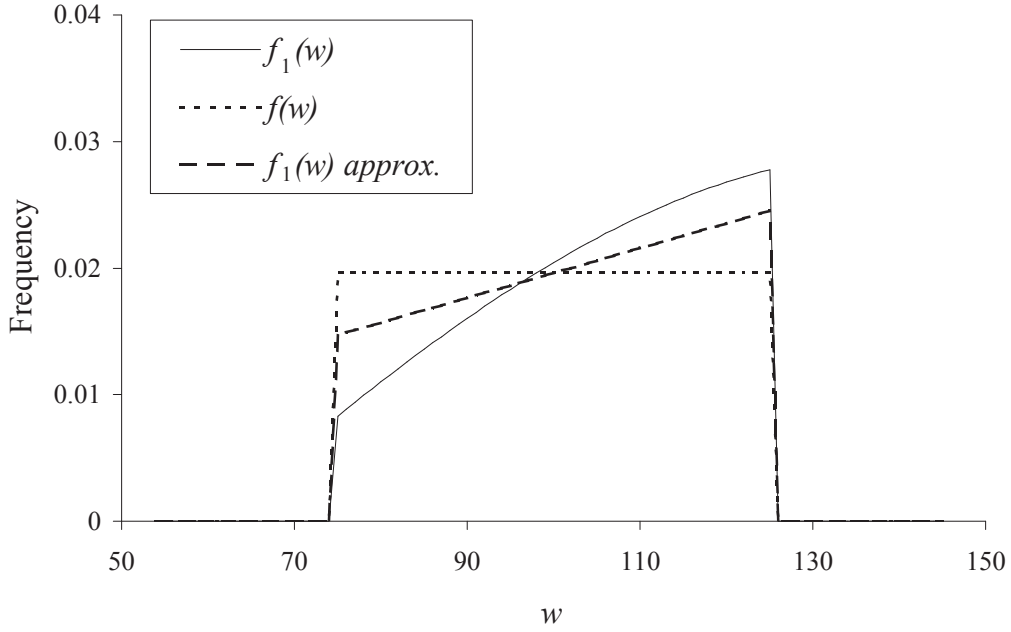


Figure 3.14: Distribution of first item $f_1(w)$ in Bin-Packing with bin size 200 and item distribution $f(w)$ being $\mathrm{U}_D(75,125)$. The Equation (3.14) approximation is included as well.

larger then the average for $f(w)$, as expected. Also, note how the average looks like damped oscillations around the infinitely-large-bin average (the Equation (3.14) estimate), trending towards it as bin size increases.

3.7 Prospect Algorithm for Bin-Packing

The Prospect function is the same as in Bin Covering, but to handle the Bin-Packing problem, the Prospect Algorithm has to be modified slightly. The modification is based on the Prospect Ratio Algorithm 3.2:

Algorithm 3.4 (Prospect Ratio Algorithm for Bin-Packing (PR-BP))

Step 1: Pick the next item and test-fit it into all active bins. If it has positive PIR for at least one of the bins, then go to Step 2. Otherwise, go to Step 3.

Step 2: Put the item into the bin with the highest PIR (ties can be resolved arbitrarily). If the selected bin gets within zone size z to be filled, then close it and activate a new empty bin. Go to Step 1.

Step 3: If no spoiled bin exists go to Step 4. Else put the item in the spoiled bin if it fits, if not close the spoiled bin and put the item into a new empty bin. Go to Step 1.

Step 4: Put the item into the fullest bin it fits into. If the item does not fit into any bin, close the fullest bin, and put the item into a new empty bin. Go to Step 1.

Note that Step 3 is needed since it is possible that a spoiled bin is not the fullest. For example, if the distribution is U_D(75,125), bin size is 300, zone size is 5, and there is a space $w = 135$ left in a bin, the bin has prospect zero since it cannot be finished within the zone with one item, and it has not enough room left for two items. On the other hand, if $w = 100$, the bin has positive prospect.

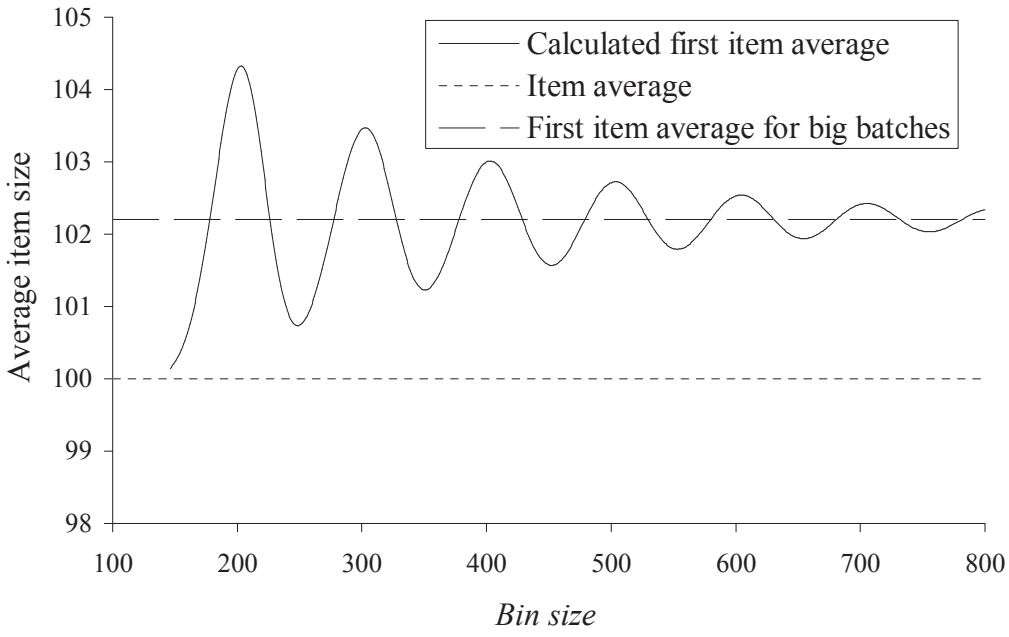


Figure 3.15: Average of first item in Bin-Packing with varying bin size for $N_D(100,15)$. Item average and first item average as the bin size goes to infinity are plotted as well.

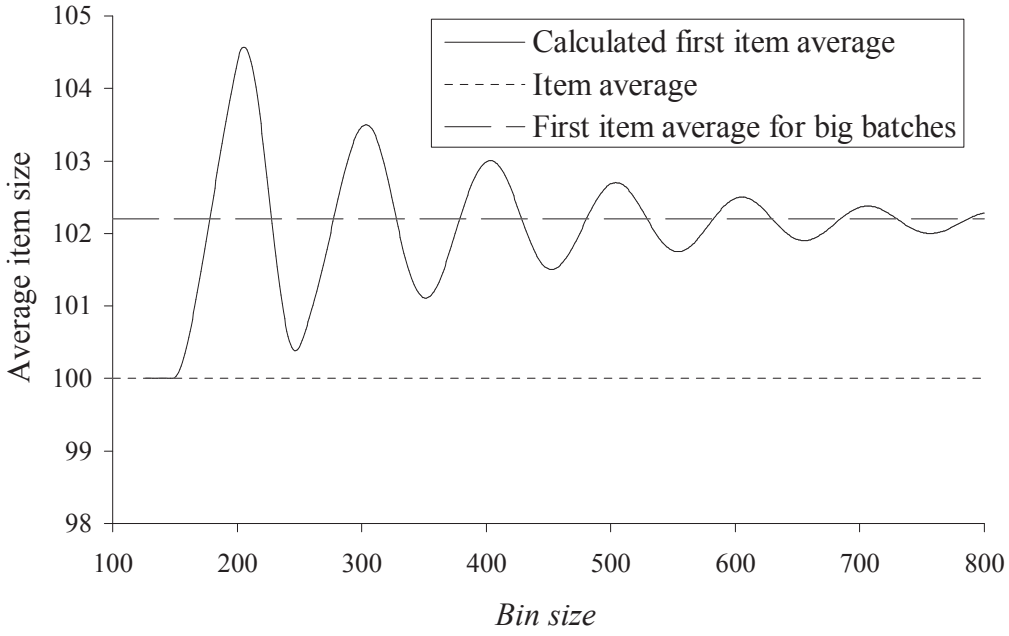


Figure 3.16: Average of first item in Bin-Packing with varying bin size for $U_D(75,125)$. Item average and first item average as the bin size goes to infinity are plotted as well.

To test this algorithm, it is simulated with items from the $N_D(100,15)$ distribution, 8 active bins, and bin size varying from 200 to 800 in increments of 20. Each simulation is run for 10000 filled bins. Two versions of the algorithm are used: PR-BP, which uses the Prospect function from Section 3.2, and PR-BP1, which uses the Prospect function from Section 3.6. Simulation results are shown in Figure 3.17. It is interesting to note that there is not an improvement in performance obtained from using the Prospect function in Section 3.6 vs. the original Bin-Covering Prospect function. The reason for this can be seen when the actual average size of the first piece in a batch is viewed in Figure 3.18. The actual first piece average is usually significantly different from the value calculated using Equations (3.15) and (3.16), and both averages are close to the distribution average of 100. The difference exists because the calculated value for the first item is only valid for the Next-Fit Algorithm, which has only one active bin; but when using the Prospect Algorithm, there are multiple active bins that each item can potentially go into.

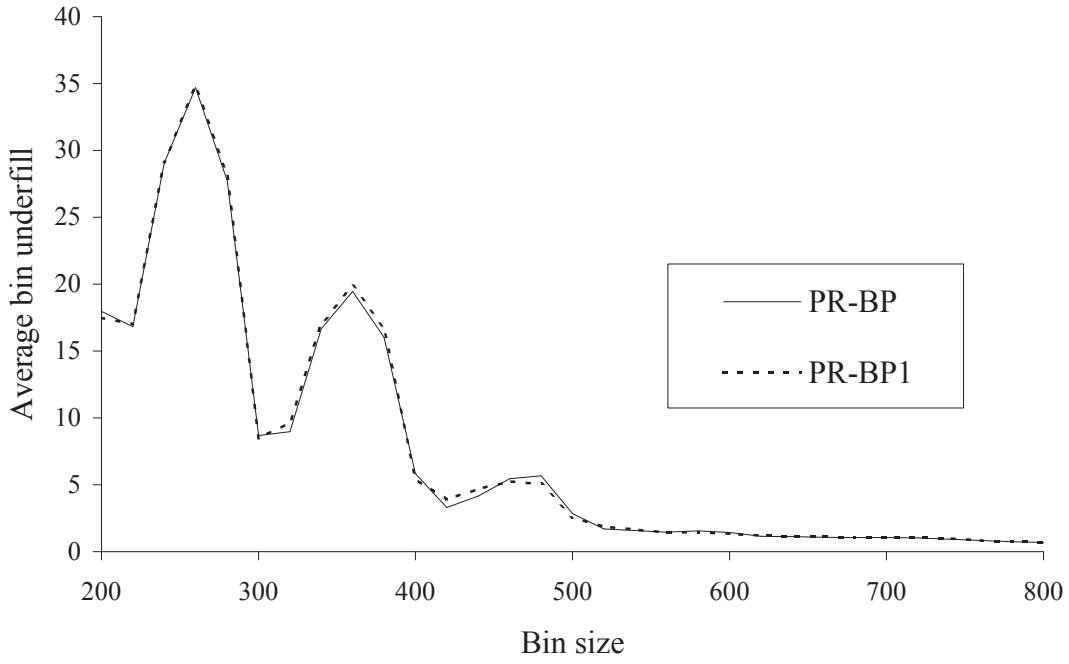


Figure 3.17: Average bin underfill in Bin-Packing. Items are from the $N_D(100,15)$ distribution, and there are 8 active bins.

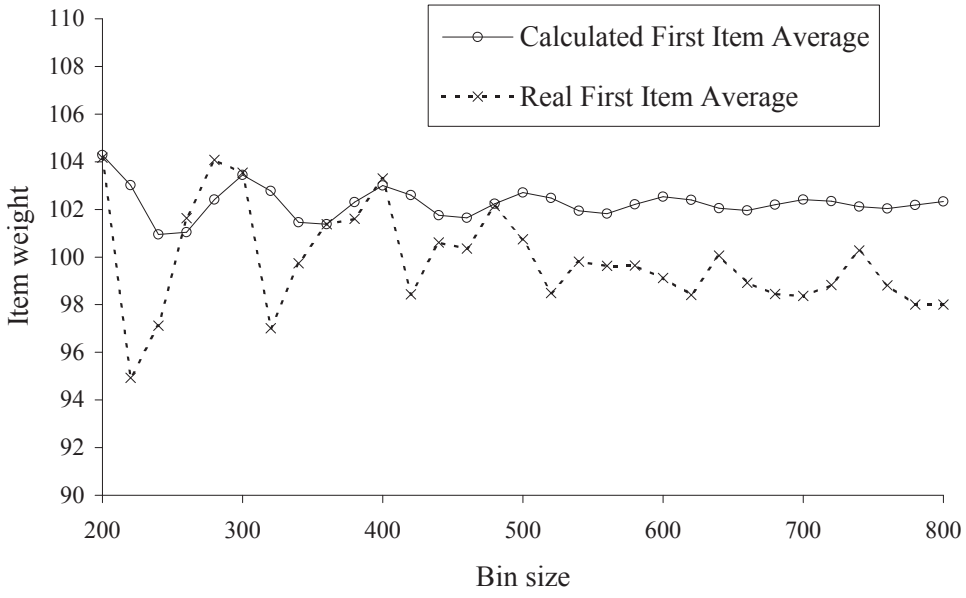


Figure 3.18: Calculated and actual first piece average in Bin-Packing using PR-BP1. Items are from the $N_D(100,15)$ distribution, and there are 8 active bins.

CHAPTER 4

EXTENSIONS TO THE PROSPECT ALGORITHM

The Prospect Algorithm can easily be modified to work on many different variants of the Bin-Covering problem. In Section 4.1, a variant of the Prospect Algorithm is introduced where items that are unsuitable for packing are rejected. In Section 4.2, a constraint on the number of items per bin is added, and in Section 4.3, the Prospect Algorithm is extended to handle items of more than one type. Finally, in Section 4.4, the Prospect Algorithm is extended to handle problems where the algorithm has more than one bin capacity to choose from.

4.1 Bin-Covering with Reject

As Figure 3.10 shows, the bin size can have a great impact on the performance of Bin-Covering algorithms, especially as the bin size becomes small and is not near an even multiple of the average item size. However, suppose that there is no need to put all items into bins. Instead, only those items that 'fit' well into bins are actually placed there, and other items are rejected. It turns out that a slightly modified version of Algorithm 3.2 works for this problem. The new algorithm requires an extra parameter, v, that acts as a *cutoff* on the PIR. No item is put into a bin whose PIR $< v$. By setting this cutoff, one is essentially setting a limit on how much one allows the value of the Prospect function for a bin to deteriorate in order to accommodate a new item.

Algorithm 4.1 (Prospect Algorithm with Reject)

Step 1: Pick the next item and test-fit it into all active bins. If the highest PIR is $\geq v$, then go to Step 2. Otherwise go to Step 3.

Step 2: Put the item into the bin with the highest PIR (ties can be resolved arbitrarily). If the selected bin gets filled, then close it and activate a new empty bin. Go to Step 1.

Step 3: Reject the item and go to Step 1.

Note that in this problem, the algorithm only generates filled bins that are within the predefined zone because it does not allow bins to be spoiled. Further, by varying z, there is a trade-off between the ratio of accepted items and the average bin overfill. To test Algorithm 4.1, two bin capacities from Figure 3.10 that have high average overfill are selected, namely, 240 and 340. The data sets are the same as in Figure 3.10. The zone size is varied from 0 to 40 in increments of 2 and v is set to 0.5. In each plot both the average overfill and the percentage of items utilized are depicted. Figures 4.1 and 4.2 show the results; note how the item utilization increases with the zone size.

In Figure 4.3 (using the same data as in Figures 4.1 and 4.2), we plot how the average overfill changes with the item utilization. From this, the trade-off between item utilization and overfill can be seen. In particular, if one wants to use more of the items, then one has to accept higher overfill. Also, note that the larger bin capacity performs better, in that it yields higher item acceptance for the same average overfill.

A suitable value of v depends both on the problem setup and on the value of z. In Figure 4.4, the cutoff, v, is varied (bin size is 240), using the same data as before in this section. This shows that a higher cutoff value can improve the performance when the item acceptance rate is low.

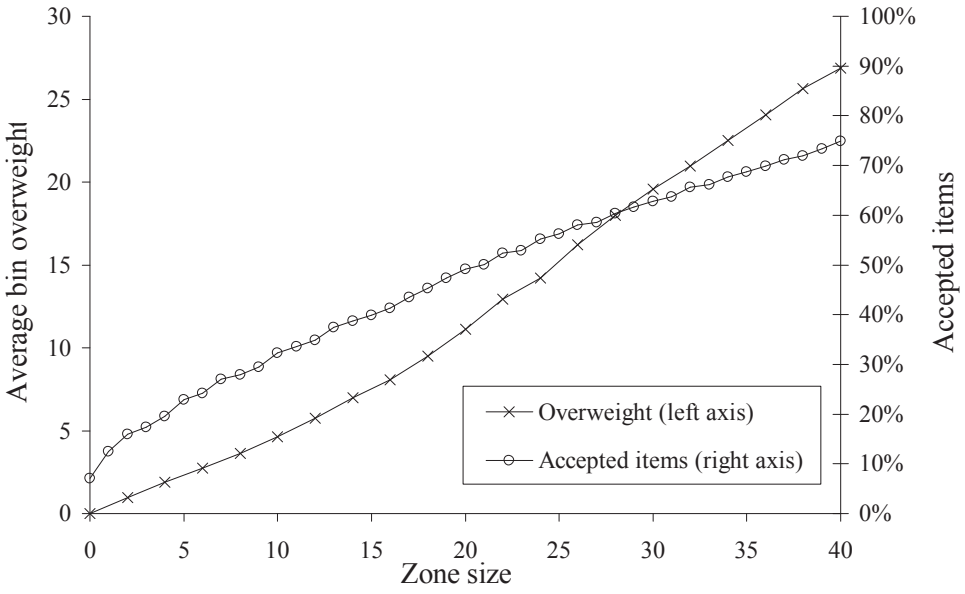


Figure 4.1: Average overfill and item utilization as a function of zone size for eight active bins, bin capacity 240, and the $N_D(100,15)$ distribution.

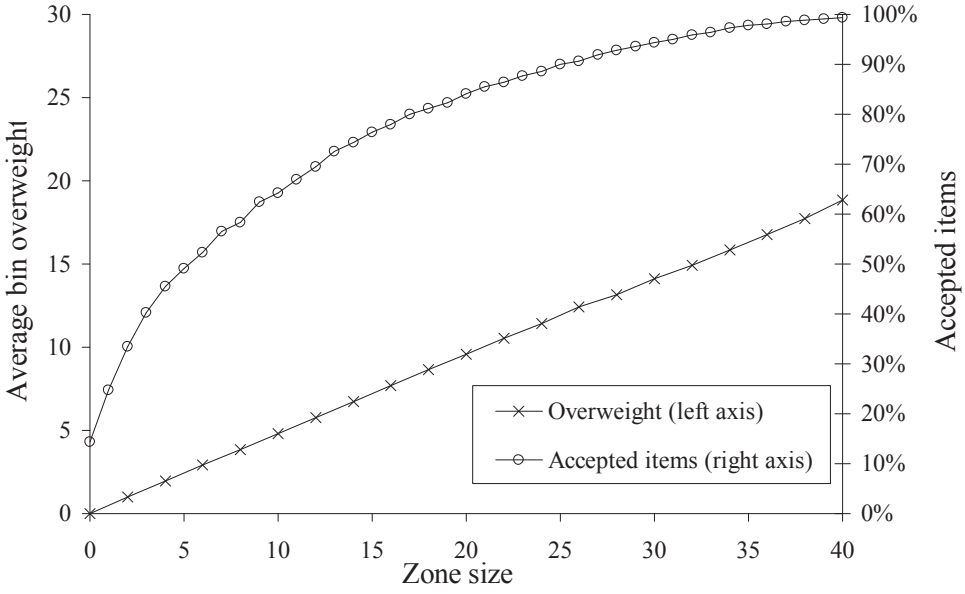


Figure 4.2: Average overfill and item utilization as a function of zone size for eight active bins, bin capacity 340, and the $N_D(100,15)$ distribution.

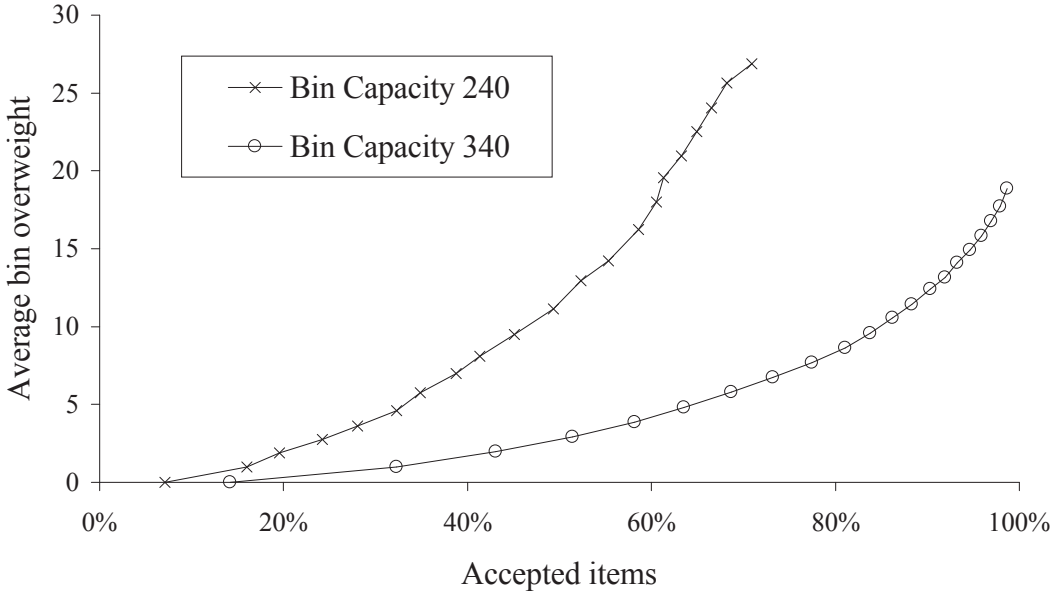


Figure 4.3: Average overfill as a function of item utilization for eight active bins, and the $N_D(100,15)$ distribution.

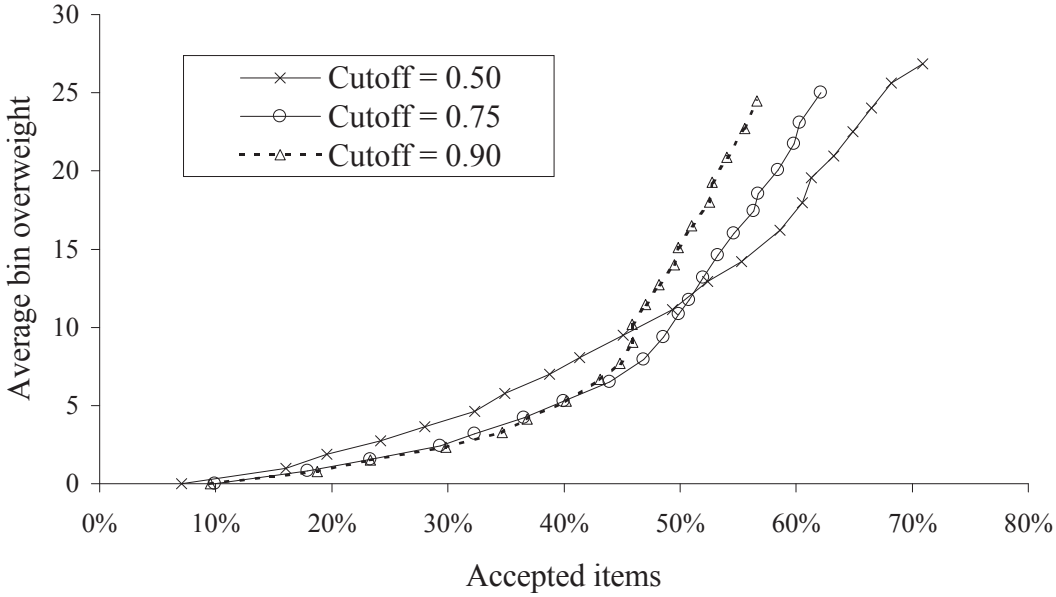


Figure 4.4: Average overfill as a function of the item utilization, with varying cutoff for eight active bins, bin capacity 240, and the $N_D(100,15)$ distribution.

4.2 Constraints on the Numbers of Items per Bin

The Prospect Algorithm can also work well if there are constraints on the number of items per bin. If the covering problem has an upper bound on the number of items per bin, then the possibility of reject usually must be allowed (unless the constraints are redundant). The reason for this is that if the algorithm is forced to use every item, then there is a positive probability that it will have to force an item into a bin, violating the constraints. This is not a problem if there is only a lower bound on the number of items per bin, since then it is easy to add items to the bin to satisfy the constraint. Algorithm 4.1 can still be used, but a modified version of Equation (3.2) must be used to calculate the prospect. The quantity c^{b} is defined as the number of items in the bin so far, and $c^{\min}$ and $c^{\max}$ are the minimum and maximum numbers of items per bin, respectively. Then the prospect is calculated by summing over all allowed numbers of items in the bin, assuming that $f_{\min} > z$, and defining $c^{\mathrm{start}} \equiv \max(1, c^{\min} - c^{\mathrm{b}})$ and $c^{\mathrm{end}} \equiv c^{\max} - c^{\mathrm{b}}$:

$$\mathrm{Pf}(w, z, c^{\mathrm{b}}, c^{\min}, c^{\max}) \equiv \begin{cases} \displaystyle\sum_{c=c^{\mathrm{start}}}^{c^{\mathrm{end}}} \sum_{i=0}^{z} f^{(c)}(w+i) & \text{if } w > 0, \\ 1 & \text{if } -z \le w \le 0 \text{ and } c^{\min} \le c^{\mathrm{b}} \le c^{\max}, \\ 0 & \text{else.} \end{cases} \tag{4.1}$$

Figure 4.5 shows simulation results for this type of problem. The data sets are the same as in Figure 3.10, the bin size is 400, there are exactly 4 items in each bin (so that $c_{\min} = c_{\max} = 4$), $z = 10$, and the number of bins is varied from 4 to 16. The reject cutoff v is 0.5. The figure shows that the average overfill does not depend on the number of active bins, but is approximately half of z. That the average overfill does not depend on the number of bins is expected since the algorithm is allowed to reject unsuitable items, ensuring that the finished bins stay within the zone (end up

within half of the zone on average). However, it is the reject rate that is affected by the number of bins; it drops with increasing number of bins.

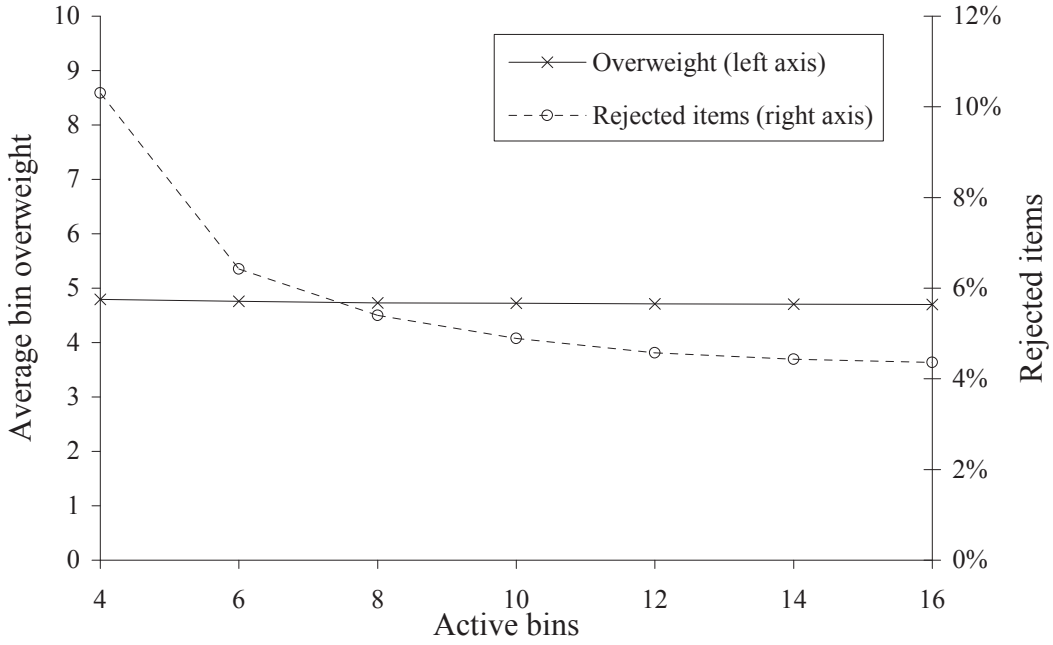


Figure 4.5: Average overfill and item utilization for different number of bins, bin capacity 400, exactly 4 items per bin, and the $N_D(100,15)$ distribution.

4.3 Many Types of Items

The Prospect Algorithm can be extended to handle items of more than one type. A practical example for when this is needed is when poultry manufacturers want to make packs of cut-up chicken where the count of each type is predefined and the minimum weight is set as well. For example, in South Africa a popular grocery product is a bag of individually quick frozen chicken pieces, typically 1 kg, 1.5 kg, or 2 kg of total weight. The customers expect to get a variety of pieces, so the manufacturers usually want to guarantee that there are at least two pieces of each type.

The Prospect Algorithm can be extended to work on this problem, but the calculation of the prospect becomes more complicated. The formula for calculating the prospect is an extension of Equation (4.1), and if there is an upper bound on the

number of items, then the algorithm must be able to reject unsuitable items for the same reason as described in the previous section. Define n as the number of different types of items, and here the following variables from Equation (4.1) are vectors of domension n instead of scalars: Number of pieces in the bin so far $\mathbf{c}^{\mathrm{b}}$, minimum number of pieces per bin $\mathbf{c}^{\min}$, and maximum number of pieces per bin $\mathbf{c}^{\max}$. The derived start and end counts are also vectors:

$$\mathbf{c}^{\mathrm{start}} = [\max(1, c_1^{\min} - c_1^b), \max(1, c_2^{\min} - c_2^b), \ldots, \max(1, c_n^{\min} - c_n^b)]$$

$$\mathbf{c}^{\mathrm{end}} = [c_1^{\max} - c_1^b, c_2^{\max} - c_2^b, \ldots, c_n^{\max} - c_n^b].$$

Let $\mathbf{c}$ be a vector indicating the number of items of each type in a bin (so that $\mathbf{c} = [c_1, c_2, \ldots, c_n]$, where c_i is the number of items of type $i \in \{1, 2, \ldots, n\}$), and define the function $f(w, \mathbf{c})$ as the probability that the specific combination of items given by $\mathbf{c}$ will sum up to w. This function can be calculated directly from the individual probability mass functions for the different item types, either by convolution, or by using the central limit theorem (similar to Equation (3.5)). The quantity $\Sigma\mathbf{c}$ is defined as $\sum_{i=1}^{n} c_i$, and p_i is the relative frequency of item type i among all items $(i = 1, \ldots, n)$. It is easy to see that the prospect that a given combination of items, $\mathbf{c}$, and weight left in bin w ends within the zone z is:

$$\mathrm{Pf}_{\mathrm{MT}}(w, z, \mathbf{c}) \equiv \left(\sum_{i=0}^{z} f(w + i, \mathbf{c}) \right) \left((\Sigma\mathbf{c})\,! \prod_{j=1}^{n} \frac{p_j^{c_j}}{c_j!} \right). \tag{4.2}$$

The first part of Equation (4.2) represents the probability that the given combination $\mathbf{c}$ has total size in the range of $[w, w+z]$, and the second part represents the probability that the given combination $\mathbf{c}$ occurs. Now the prospect calculation can be formulated by summing over all relevant count combinations from $\mathbf{c}^{\mathrm{start}}$ to $\mathbf{c}^{\mathrm{end}}$:

$$\mathrm{Pf}(w, z, \mathbf{c}^{\mathrm{b}}, \mathbf{c}^{\min}, \mathbf{c}^{\max}) = \begin{cases} \displaystyle\sum_{\mathbf{c}=\mathbf{c}^{\mathrm{start}}}^{\mathbf{c}^{\mathrm{end}}} \mathrm{Pf}_{\mathrm{MT}}(w, z, \mathbf{c}) & \text{if } w > 0, \\ 1 & \text{if } -z \le w \le 0 \text{ and } \mathbf{c}^{\min} \le \mathbf{c}^{\mathrm{b}} \le \mathbf{c}^{\max} \\ 0 & \text{else.} \end{cases} \tag{4.3}$$

The condition $\mathbf{c}^{\min} \le \mathbf{c}^{\mathrm{b}} \le \mathbf{c}^{\max}$ ensures that $c_i^{\min} \le c_i^{\mathrm{b}} \le c_i^{\max}$ for all i, and hence only bins with allowed numbers of items by type get finished. Note that the sum $\sum_{\mathbf{c}=\mathbf{c}^{\mathrm{start}}}^{\mathbf{c}^{\mathrm{end}}}$ runs through all combinations of $\mathbf{c}$ between $\mathbf{c}^{\mathrm{start}}$ and $\mathbf{c}^{\mathrm{end}}$. The number of loops required for this is given by the following formula:

$$\prod_{i=1}^{n} \left(c_i^{\mathrm{end}} - c_i^{\mathrm{start}} + 1\right). \tag{4.4}$$

This quantity can quickly become large, so if the differences between the c_i^{start}'s and c_i^{end}'s become too large, the model becomes unpractical because of long run time. This is not a significant problem in practice because if there are large differences between the c_i^{start}'s and c_i^{end}'s, that usually means that the constraints are redundant and can be ignored.

Figure 4.6 shows simulation results for this type of problem. The data set used is from the South African poultry industry, it is summarized in Table 4.1, and described fully in Appendix A.21. The bin size is 1500, and there are 2–5 items of each type allowed in each bin. Similar to Figure 4.5, Figure 4.6 shows that the average overfill does not depend greatly on the number of active bins, which is expected since the algorithm is allowed to reject unsuitable items. The figure also shows that the rejection rate drops with increasing number of active bins.

4.4 Multiple Bin Capacities

This variation of the Bin-Covering problem comes from the Norwegian Salmon Farming industry. There it is a common practice to put fresh or frozen salmon into packs,

Table 4.1: Summary of the chicken distribution.

Type	Count	Frequency	Average	StdDev	Min	Max
Breasts	398	25%	326.83 g	38.79 g	217 g	441 g
Drums	398	25%	97.00 g	14.65 g	63 g	143 g
Thighs	398	25%	204.31 g	33.02 g	123 g	339 g
Wings	398	25%	85.96 g	17.40 g	40 g	154 g
Total	1592	100%	178.52 g	101.24 g	40 g	441 g

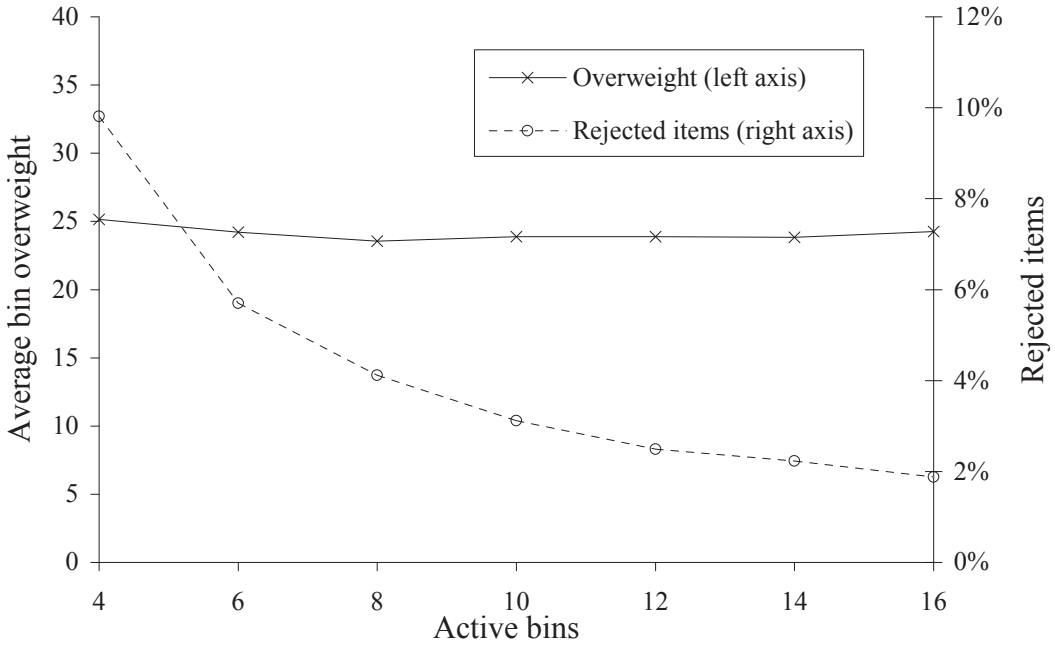


Figure 4.6: Average overfill and item utilization for different numbers of bins, bin capacity 1500, the Chicken distribution, and 2–5 items of each type allowed.

where the guaranteed weight is rounded down to the nearest kilogram (or the nearest half kilogram). The packs typically have a range of 3–5 allowed weights, for example, 24, 25 or 26 kg. This problem is a version of the variable bin-size problem mentioned in Section 2.2 on Page 20. The seller can save money by targeting the packs to be close to (but over) the allowed pack weights. The Prospect Algorithm can easily handle this extension of the problem. In particular, a zone above each allowed bin capacity is defined. We focus on the case where the smallest item used, $f_{\min}$, is larger than the difference between the largest and smallest allowed bin capacities. Then it

is sufficient to modify Equation (3.2). If this condition is not true, then a correction term similar to the one in Equation (3.1) is needed. Let w_i be the empty space corresponding to bin capacity i, and let $\mathbf{w} = [w_1, w_2, \ldots, w_n]$, where n is the number of different bin capacities allowed. The vector $\mathbf{w}$ is sorted by increasing size. Then the Prospect function can be calculated by summing up the individual prospects of each different bin capacity:

$$\mathrm{Pf}(\mathbf{w}, z) = \sum_{j=1}^{n} \sum_{i=0}^{z} \mathrm{Pf}(w_j + i). \tag{4.5}$$

In Figure 4.7, the Multiple Bin Capacity (MBC) version of the Prospect Algorithm is compared to the original version. The data was logged at a Norwegian farmed salmon plant for fresh whole salmon. The resolution of the data was 5 grams and therefore we divided by 5 to make the resolution equal to a standardized resolution of 1. The weight range of the salmon was between 2.9 and 3.7 kg, or 580 and 740 after the division. The pack size (bin capacity) is allowed to be 24, 25, or 26 kg, which corresponds to 4800, 5000, and 5200, respectively, after the division. The data average is 652.7, and the standard deviation is 43.02. This distribution is called FS(653,43) and it is described in Appendix A.20. The first three bars in Figure 4.7 show the average overfill given by the regular PR Algorithm (Algorithm 3.2) for the three allowed bin capacities, and the fourth shows the average overfill for the MBC algorithm. The simulation results show that there is considerable overfill savings possible by allowing multiple bin capacities.

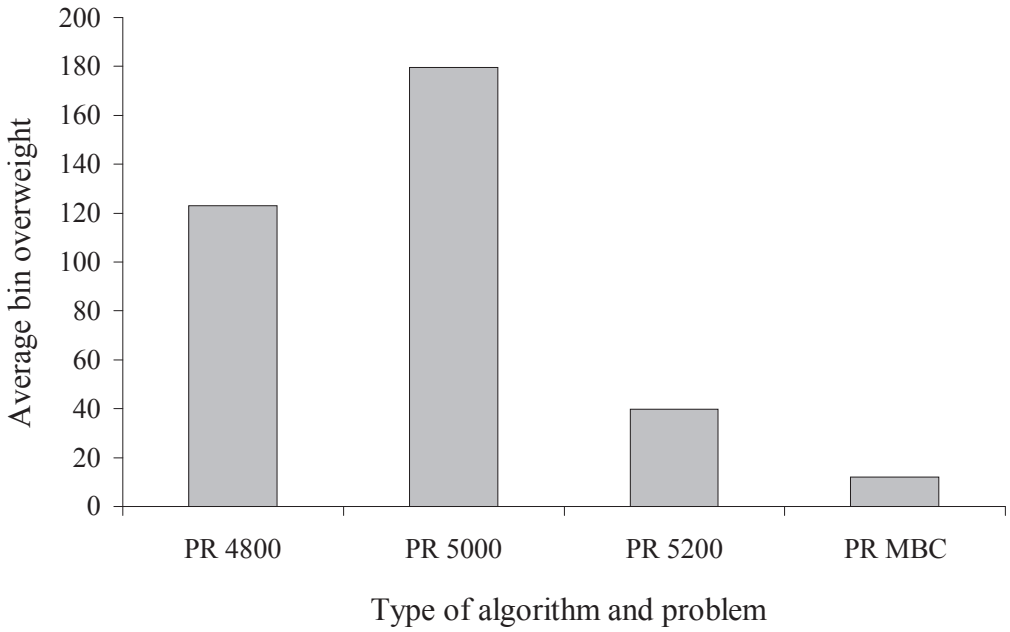


Figure 4.7: Comparison of regular and multiple bin capacity algorithms.

CHAPTER 5

PERFECT PACKING ANALYSIS

5.1 Introduction

The discrete Perfect Packing theory [31, 32] discussed on Page 17 states that when packing items from a discrete distribution F (with rational-valued probabilities) into bins, the problem can be categorized into one of three classes depending on the growth rate of the space wasted by an optimal packing algorithm. The wasted space is defined as the total sum of empty space present in all unfinished bins, and it is assumed that no bin is finished unless it gets filled exactly. The space wasted by an optimal algorithm will either grow as $\Theta(n)$, $\Theta(\sqrt{n})$, or $O(1)$, where n is the number of items processed [33] (formal definitions in Section 2.1, Page 7). If the wasted space grows as $\Theta(\sqrt{n})$ or $O(1)$, then the problem is said to be perfectly packable. Further, if the wasted space grows as $O(1)$, the problem is called *bounded space*.

The above theory applies to both Bin-Packing and Bin-Covering problems, and for the same item distribution and bin size, Bin-Packing and Bin-Covering will have the same optimal wasted space growth rate. It should be noted that in these problems, there is no upper bound on the number of active bins, so unless the problem is bounded space, the number of active bins in an optimal packing algorithm will grow infinitely large as more and more items are processed. An algorithm achieves the optimal waste rate if its wasted space has the same growth rate as the optimal wasted space.

5.2 The Sum-of-Squares algorithm

Csirik et al. [38, 43, 44] have presented an On-Line Bin-Packing algorithm called Sum-of-Squares (SS) (see Page 18 of Section 2.2), and in this section it is described.

The description is short; the above references provide a full description.

The basic SS algorithm works on discrete item distributions where all item sizes and the bin size are integer-valued. The basic algorithm works as follows. Let B be the bin size and let $n(w)$ be the number of active bins whose content totals w, where $0 < w < B$ (i.e., these bins have a "gap" of size $B - w$). Call the vector $\mathbf{n} = [n(1), n(2), \ldots, n(B-1)]$ the profile of the active bins, and the Sum-of-Squares of the active bins profile is then defined as: $SS(\mathbf{n}) = \sum_{w=1}^{B-1} n(w)^2$. The bin selection algorithm is:

Definition 5.1 (Sum-of-Squares objective)

Place the next item so as to minimize $SS(\mathbf{n}) = \sum_{w=1}^{B-1} n(w)^2$ for all the active bins.

The motivation behind the SS algorithm is to try to keep a small and approximately equal number of bins at each gap level, and thereby maximize indirectly the probability that a new item will fill a bin exactly. By keeping an equal number of bins at each level, the number of open levels is kept high, and that increases the probability that a bin is finished exactly since when a new item arrives, there is a relatively large selection of available gap levels to choose from. When deciding where to place an item of size s, it is not necessary to explicitly calculate $\sum_{w=1}^{B-1} n(w)^2$, since at each bin selection at most two entries of the sum change, $n(w)$ and $n(w+s)$. So assuming that $w > 0$ (the item is added to an already active bin) and $w + s < B$ (the item does not fill the bin) and denoting $\mathbf{n}$ and $\mathbf{n}'$ as the active bin profiles before and after the

item is added respectively, the change in Sums-of-Squares is calculated as follows:

$$\begin{aligned} SS(\mathbf{n}') - SS(\mathbf{n}) &= \sum_{w=1}^{B-1} n'(w)^2 - \sum_{w=1}^{B-1} n(w)^2 \\ &= n'(w)^2 + n'(w+s)^2 - (n(w)^2 + n(w+s)^2) \\ &\quad \text{(all other entries cancel out)} \\ &= (n(w)-1)^2 + (n(w+s)+1)^2 - (n(w)^2 + n(w+s)^2) \\ &= n(w)^2 - 2n(w) + 1 + n(w+s)^2 + 2n(w+s) + 1 \\ &\quad -(n(w)^2 + n(w+s)^2) \\ &= 2n(w+s) - 2n(w) + 2. \end{aligned} \tag{5.1}$$

Adding the cases when an item is added to a new bin ($w = 0$) and a bin is being filled ($w + s = B$), the bin selection to satisfy the Sums-of-Squares objective (5.1) is as follows:

$$\min_{w \in \{0,1,\ldots,B-s\}} \begin{cases} 2n(s) + 1 & \text{if } w = 0, \\ 2n(w+s) - 2n(w) + 2 & \text{if } w \in \{1, 2, \ldots, B-s-1\}, \\ -2n(w) + 1 & \text{if } w = B - s. \end{cases} \tag{5.2}$$

Basic SS is not guaranteed to achieve perfect packing, unless there is added logic to prevent the algorithm from creating bins with dead-end gap size. Dead-end gap size is defined as a gap size that is impossible to fill completely. For example, if the bin size is 6, and the distribution has two item sizes, 2 and 3, then a gap size of 1 is a dead-end, since a bin with 1 left cannot be filled exactly with an item of size 2 or 3. A modified algorithm, *SS'*, avoids dead-end gap sizes by keeping track of possible dead-end levels, and instead of creating a dead-end bin, it creates a new bin. How the dead-end levels are kept track of is not explained in detail in [38, 43, 44], except that the set of dead-end levels is initially empty and is recomputed each time a new item with a size that has not been seen before arrives. Note that the Prospect function can be used to determine dead-end levels, since if $\mathrm{Pf}(w, 0) = 0$, then w is a dead-end level. This is used in the simulations for this chapter to determine dead-end levels.

5.3 Prospect Algorithms for Perfect Packing

In this section, two different ways of adapting the Prospect Algorithm to Perfect Packing are explored. First the Prospect Algorithm is modified so that the number of active bins is not bounded. The differences from the original algorithm (see Page 31) are:

1. When an item will not fit into any of the active bins without spoiling, a new active bin is created and the item is put into that one.

2. If the item fills a bin exactly, then that bin is released and the number of active bins is reduced.

This algorithm has been tested and clear evidence that its waste rate is suboptimal has been found. In cases where the optimal waste rate is bounded, the Prospect Algorithm has been observed to have either bounded space, $\Theta(\sqrt{n})$, or $\Theta(n)$ waste rate. This is not surprising since the Prospect Algorithm does not take into account how many active bins are at any given gap level. An example is shown in Figure 5.1, where a simulation compares the Prospect and the SS' Algorithms. The simulation is run for 100,000 finished bins, and the average number of active pins for each finished 1000 bins is plotted. The number of active bins grows linearly ($\Theta(n)$ waste rate) for the Prospect Algorithm and ends at having more than 12,000 active bins on average. The SS' algorithm, however, has constant waste rate, and it has 160.3 active bins on average during the whole simulation.

A simple change in the Prospect Algorithm seems to fix this, namely, to divide the Prospect function $\mathrm{Pf}(w, 0)$ by $(1 + n(w))$ (note that since bins are only allowed to be filled exactly, the target zone size, z, is 0). This modified function is called the *Count-Prospect* function (CPf):

$$\mathrm{CPf}(w) = \frac{\mathrm{Pf}(w, 0)}{1 + n(w)}. \tag{5.3}$$

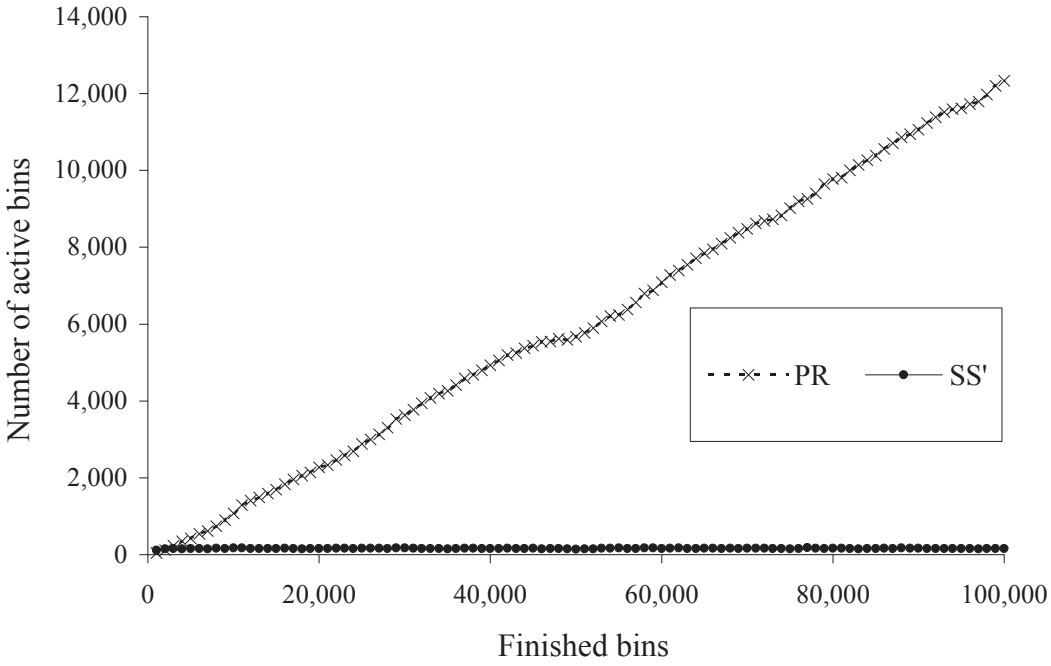


Figure 5.1: Number of active bins as a function of finished bins for the Prospect (PR) and SS' algorithms. Bin size is 480 and the $N_D(100,15)$ distribution is used.

The resulting algorithm is called the Count Prospect Algorithm (*CPR*). Simulations indicate that this algorithm appears to have optimal waste rate. The results of such simulations are shown in Figures 5.2–5.5 of the next section.

In Csirik et al. [43], it is noted that if in the SS objective function of Definition 5.1, each entry is multiplied with a function f that depends only on w and/or B, the resulting algorithm that places items so as to minimize the new objective function also achieves perfect packing (as the SS' algorithm does). We assume that the function f is strictly positive and bounded for all bin states that are not dead-end. The modified algorithm is denoted SS_f and its bin selection is defined as:

Definition 5.2 (SS_f objective)

Place the next item so as to minimize $SS_f(\mathbf{n}) = \sum_{w=1}^{B-1} f(w,B)n(w)^2$ for all the active bins.

Using a similar derivation as for (5.1), it can be easily shown that the way for the bin selection to satisfy the SS_f objective is as follows:

$$\min_{w\in\{0,1,\ldots,B-s\}} \begin{cases} f(s,B)(2n(s)+1) & \text{if } w=0, \\ f(w+s,B)(2n(w+s)+1) & \\ \quad -f(w,B)(2n(w)-1) & \text{if } w\in\{1,2,\ldots,B-s-1\}, \\ -f(w,B)(2n(w)-1) & \text{if } w=B-s. \end{cases} \tag{5.4}$$

This leads to the idea of multiplying (or dividing) by some form of the Prospect function so that inferior gap levels are penalized in the SS algorithm. An inferior gap level is a gap level that is relatively unlikely to get filled quickly to the target bin size.

We investigated several different functions f based on the Prospect function, $\mathrm{Pf}(w,0)$, namely, $1-\mathrm{Pf}(w,0)$, $1-\mathrm{Pf}(w,0)^2$, $(1-\mathrm{Pf}(w,0))^2$, $(1-\mathrm{Pf}(w,0))^{\frac{1}{2}}$, $-\ln(\mathrm{Pf}(w,0))$, $e^{-\mathrm{Pf}(w,0)}$, $1/\mathrm{Pf}(w,0)$, and $1/\mathrm{Pf}(w,0)^{\frac{1}{2}}$. We compared the performance of the SS_f algorithm with these functions using simulations. The distributions N_D(100,10), N_D(100,15), N_D(100,20) and U_D(75,125) are used with bin sizes $[600, 620, \ldots, 800]$. The bin size is kept at 600 or above to ensure that all the simulations are perfectly packable, i.e., the number of active bins does not explode. The performance of each function is simulated for the four distributions, and we compare how many active bins are used on average during the simulations. Each simulation is run for 20,000 finished bins, and the average number of open bins logged, but the first 10,000 bins are ignored for warmup. Dead-end gap levels are not considered, i.e., no bin gap levels are allowed that have $\mathrm{Pf}(w,0)=0$. This also ensures that one never divides by zero in the modifying functions. The average numbers of open bins in the simulations are listed in Table 5.1. The best performance for each simulation is shown in bold.

The simulations show that the modifications $1/\mathrm{Pf}(w,0)$ and $1/\mathrm{Pf}(w,0)^{\frac{1}{2}}$ work best. The modification $1/\mathrm{Pf}(w,0)$ has the best overall performance but does not always improve on the original SS algorithm (the U_D(75,125) distribution) as the

Table 5.1: Comparison of average number of open bins using different prospect modification functions for the SS algorithm.

	N_D(100,10)	N_D(100,15)	N_D(100,20)	U_D(75,125)	Overall
$1 - \mathrm{Pf}(w,0)$	74.05	29.54	30.47	9.14	35.80
$1 - \mathrm{Pf}(w,0)^2$	74.89	29.93	30.90	9.22	36.24
$(1 - \mathrm{Pf}(w,0))^2$	73.73	29.53	30.40	9.13	35.70
$(1 - \mathrm{Pf}(w,0))^{\frac{1}{2}}$	73.76	29.66	30.55	9.14	35.78
$-\ln(\mathrm{Pf}(w,0))$	70.54	29.67	30.23	9.18	34.91
$e^{-\mathrm{Pf}(w,0)}$	73.84	29.53	30.40	9.13	35.72
$1/\mathrm{Pf}(w,0)$	59.27	**11.81**	**13.16**	9.72	**23.49**
$1/\mathrm{Pf}(w,0)^{\frac{1}{2}}$	**53.98**	17.28	17.29	**9.07**	24.41
SS'	85.19	33.41	34.77	9.64	40.75

$1/\mathrm{Pf}(w,0)^{\frac{1}{2}}$ does. These two modifying functions, $1/\mathrm{Pf}(w,0)^{\frac{1}{2}}$ and $1/\mathrm{Pf}(w,0)$, are used for performance comparisons in the next section, labeled SS'/P½ and SS'/P, respectively.

5.4 Performance Comparison

In this section, the SS', CPR, SS'/P½ and SS'/P algorithms are compared using simulation. The simulations are set up as in the previous section except they are run longer, and batch means are calculated. Each simulation is run for 210,000 finished bins. The first 10,000 bins are ignored for warmup, and then average numbers of active bins for each batch of 2,000 finished bins, or 100 batches in total, are logged. The warmup time is much longer than in the simulations of Section 3.5 because in this case the number of open bins is much larger and the simulation needs more time to achieve steady-state. Using this data, the 95% confidence interval for the expected average number of active bins is calculated and used to compare the algorithms. The slope and its 95% confidence interval of a linear regression line for the series of 100 batch means is calculated as well. The slope and its confidence interval are used as simple statistical checks to indicate if perfect packing is being achieved.

Figures 5.2–5.5 show the results of the simulations. There are eleven simulations for each algorithm in each figure, or 44 total simulations per figure, the total number of simulations is therefore $44 * 4 = 176$. Note that the of 95% confidence intervals are not included in the plots, but they are listed in Appendix D, along with the confidence interval associated with each linear regression. The slope of the linear regression is significant in 8 out of the 176 simulations, or 4.6% of the cases, as expected given that the confidence level is 95%. We therefore conclude that the algorithms are all bounded space, and therefore achieve perfect packing.

The simulations show that the three Prospect Algorithms are usually more efficient than the SS' algorithm. This is not surprising since the prospect algorithms utilize the extra information given in the distribution, but the SS' algorithm does not. It is interesting to see that the difference in performance is much less for the uniform distribution, than for the normal distribution. This phenomenon can be explained by the fact that the pmf of a (discrete) uniform distribution is simpler than the pmf of a (discrete) normal distribution, and hence there is less information in the uniform distribution that can be utilized by the prospect algorithms, making their performance closer to the performance of the SS' algorithm. CPR and SS'/P½ always perform better than SS'. But SS'/P relative performance is more unstable – it has the best performance of all in Figures 5.3 and 5.4, the third best in Figure 5.2, and the worst in Figure 5.5. The reason for this phenomenon is not clear, but it is consistent with the simulations in the preceding section; see Table 5.1.

5.5 Improved Prospect+ Algorithm

In this section we examine how the Prospect Algorithm can be improved using the modified Count-Prospect function (5.3) from Page 64. To test the improvements and compare them to the original Prospect Algorithm, the same simulation setup is used as in Section 3.5. For each item distribution the bin size is varied from 200 to 800

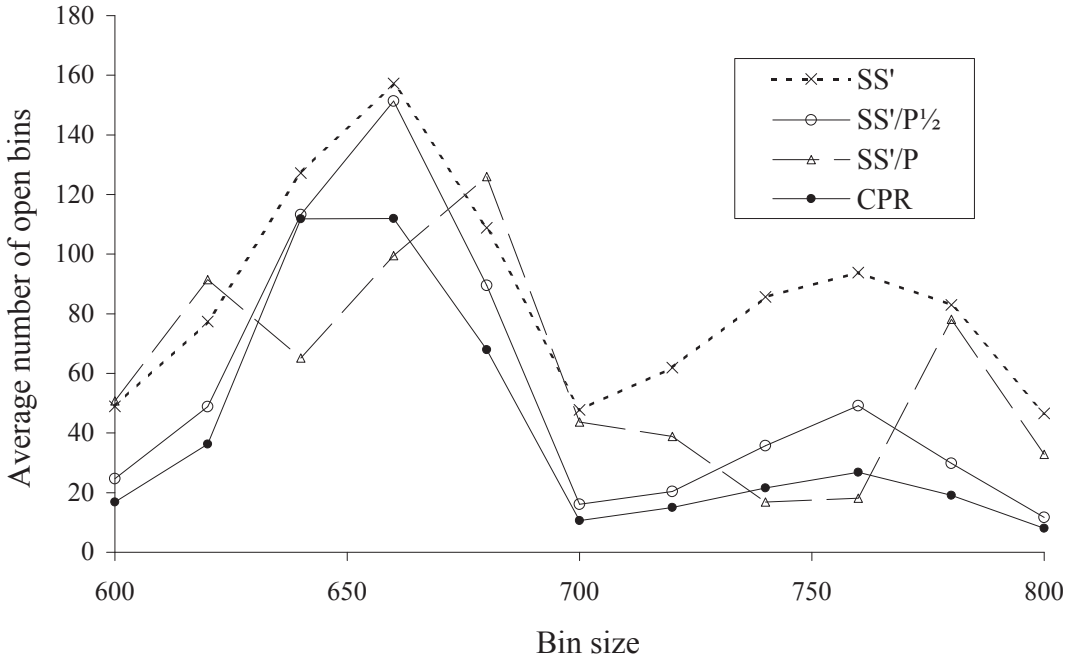


Figure 5.2: Comparison of the performance of the SS', Prospect-SS', and Count Prospect Algorithms. Bin size is varied and the $N_D(100,10)$ distribution is used.

in increments of 20, for a total of 31 simulations. Each simulation starts up with a warmup period lasting until 1,000 bins have been filled, and after that the simulation is run until $20{,}000 * 30 = 600{,}000$ bins have been generated. The bins are grouped into 30 batches of 20,000 bins, and the average overweight is calculated for each batch. This is then used to calculate the average overweight and standard deviation of the overweight. From these a 95% confidence interval for the average overfill is calculated. There are eight active bins. The optimal zone size for each simulation is determined with the Optimal Zone Search Algorithm 3.3 on Page 38.

CPf(w) does not transfer well to the original problem in which the number of active bins is limited, because the probability of two bins having exactly the same position can be low — so low that the correction factor $\frac{1}{1+n(w)}$ is usually just 1. To counter this, a simple change suffices to get an improved version of the Prospect Algorithm, namely, relaxing the requirement that bins must be at exactly the same location for the correction factor to apply. Define $n(w, z)$ as the number of active bins

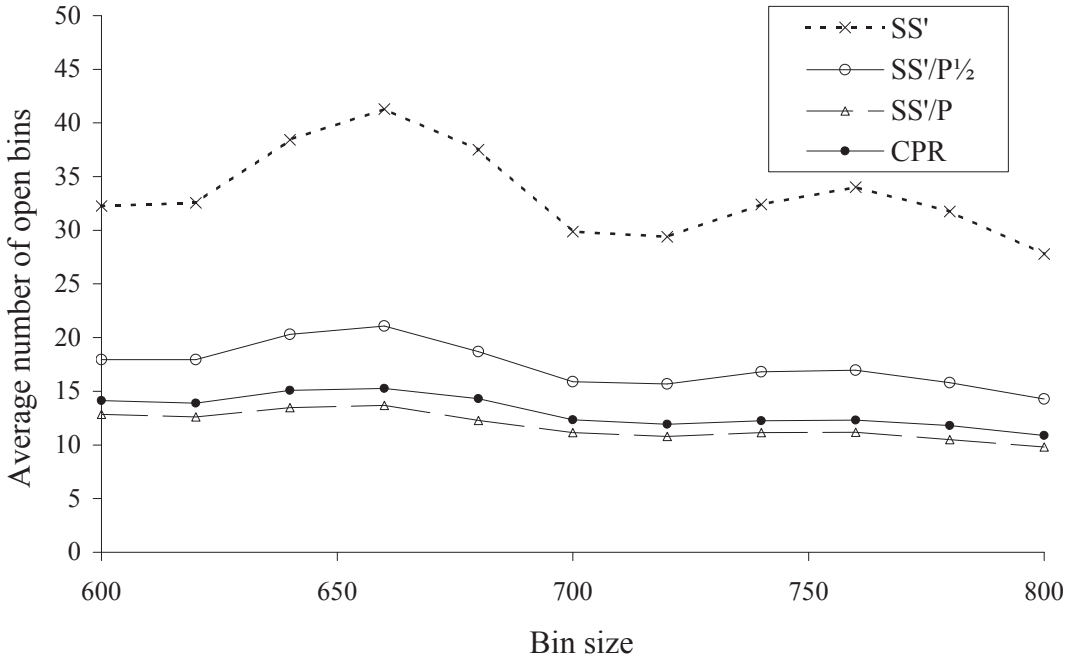


Figure 5.3: Comparison of the performance of the SS', Prospect-SS', and Count Prospect Algorithms. Bin size is varied and the $N_D(100,15)$ distribution is used.

whose contents are in $[B-w-z, B-w+z]$ (i.e., bins that have a "gap" within $w-z$ to $w+z$). Note that the zone is *two-sided* as opposed to the one-sided zone used in the Prospect Algorithm. The reason for this modification is that it usually works better than the one-sided zone of $[B-w, B-w+z]$. The entire set of simulations was run twice, once for the one-sided zone and once for the two-sided zone. A 95% confidence interval for the average overfill was calculated for each run. If the confidence intervals for the two versions do not overlap, then the one with the lower overfill is declared winner; if they overlap, then we declare a tie. The results are summarized in Table 5.2 below. The two-sided zone has better performance in 38.7% of the simulations, but the one-sided zone in only 14.5%. Therefore we select the two-sided zone as the one to implement.

The following $\mathrm{CPf}(w,z)$ is then used in the Prospect Algorithm:

$$\mathrm{CPf}(w,z) = \frac{\mathrm{Pf}(w,z)}{1+n(w,z)}. \tag{5.5}$$

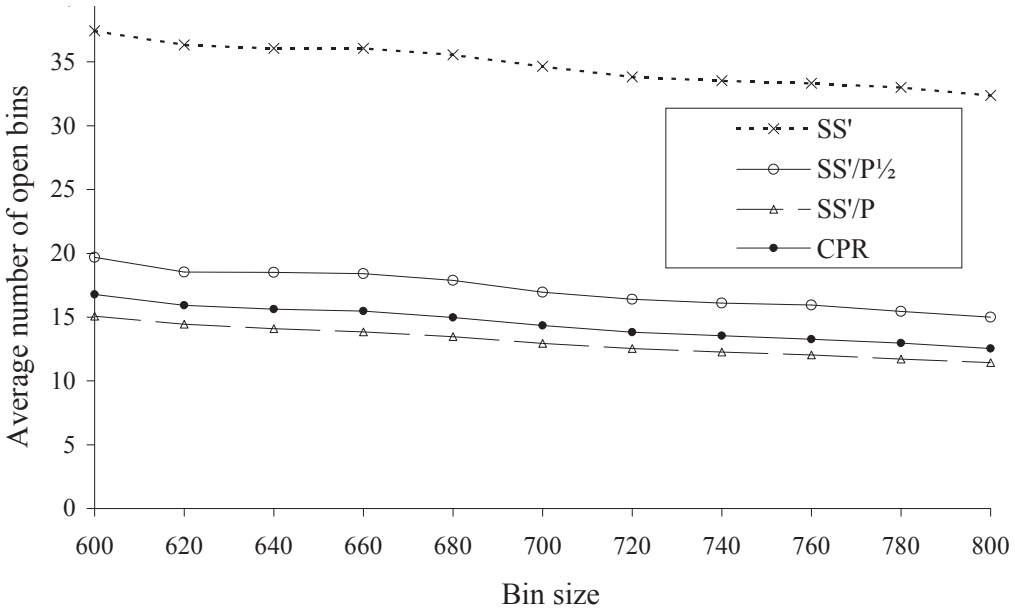


Figure 5.4: Comparison of the performance of the SS', Prospect-SS', and Count Prospect Algorithms. Bin size is varied and the $N_D(100,20)$ distribution is used.

This modified algorithm is called the *Prospect+* Algorithm, or *PR+* for short. Figures 5.6–5.9 show the results of the simulations, comparisons of the 95% confidence intervals are summarized in Table 5.3, and the full comparisons are listed in Appendix C. The comparison is not clear cut, but there seems to be a pattern in which PR performs better for small bin sizes and when the average overfill is high, but PR+

Table 5.2: "Statistical winners" comparison of one- and two-sided zones in the $n(w, z)$ function of Equation (5.5).

	Winner		
Distribution	One-sided	Tie	Two-sided
$N_D(100,10)$	5	12	14
$N_D(100,15)$	7	10	14
$N_D(100,20)$	3	20	8
$U_D(75,125)$	3	16	12
Total	18 (14.5%)	58 (46.8%)	48 (38.7%)

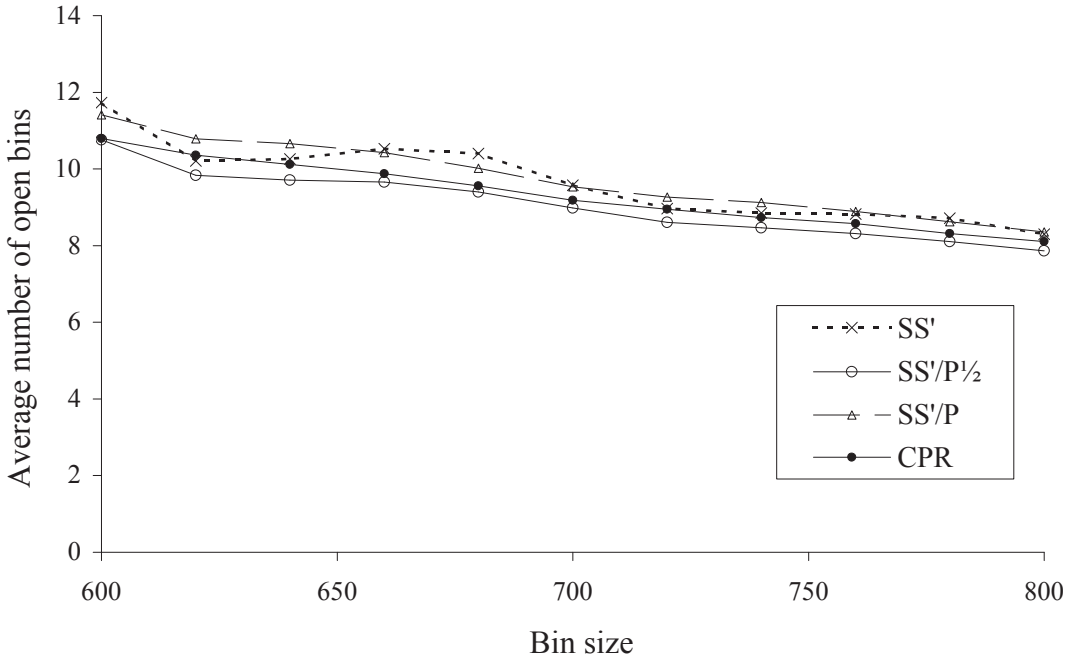


Figure 5.5: Comparison of the performance of the SS', Prospect-SS', and Count Prospect Algorithms. Bin size is varied and the $U_D(75,125)$ distribution is used.

does better when bin size increases and average overfill drops. Moreover, PR+ outperforms PR in about 70% of cases, as shown in Table 5.3. Thus PR+ is the preferred version unless the average overfill is very high, or the bin size relatively small. The difference is not great, but significant.

Table 5.3: "Statistical winners" summary of the PR and PR+ Algorithms.

	Winner		
Distribution	PR	Tie	PR+
$N_D(100,10)$	11	7	13
$N_D(100,15)$	3	3	25
$N_D(100,20)$	4	2	25
$U_D(75,125)$	1	6	24
Total	19 (15.3%)	18 (14.5%)	87 (70.2%)

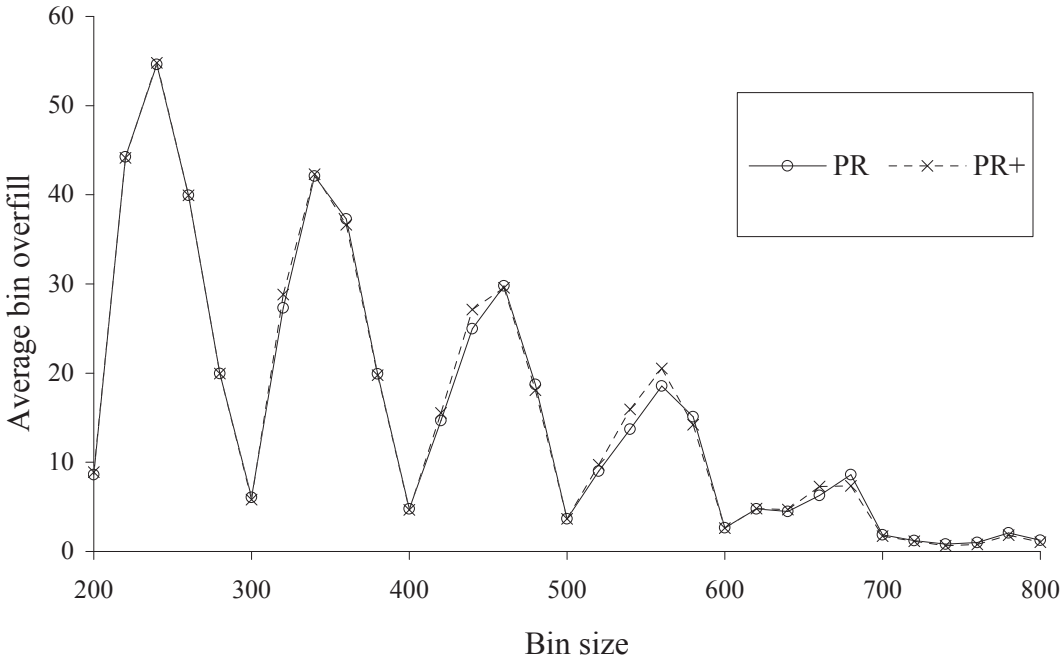


Figure 5.6: Comparing PR and PR+ algorithms for eight active bins, bin size from 200 to 800, and the $N_D(100,10)$ distribution.

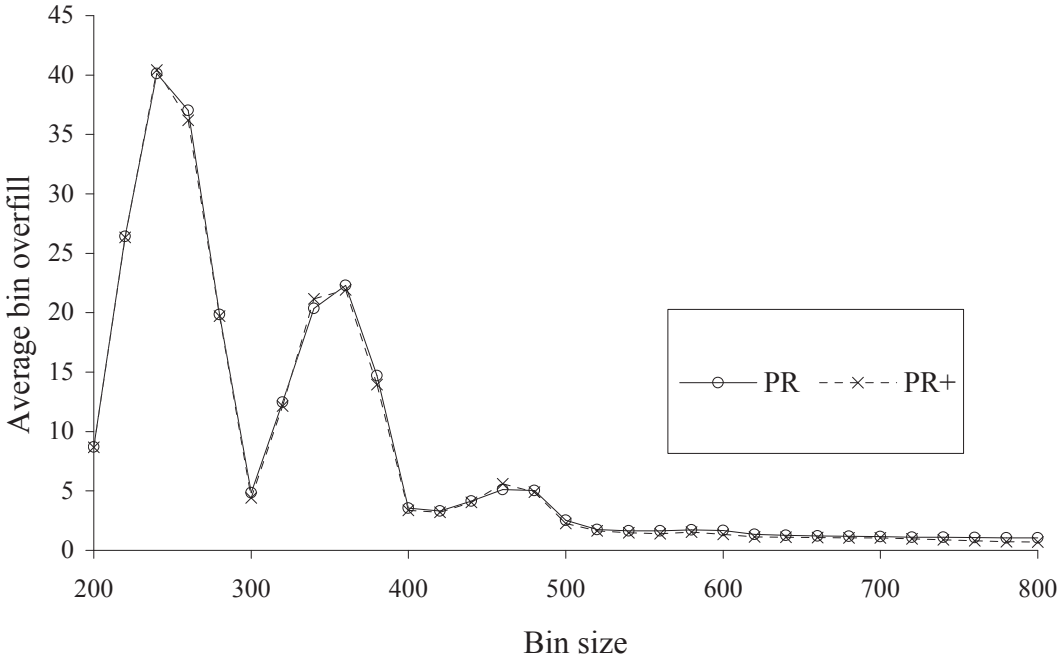


Figure 5.7: Comparing PR and PR+ algorithms for eight active bins, bin size from 200 to 800, and the $N_D(100,15)$ distribution.

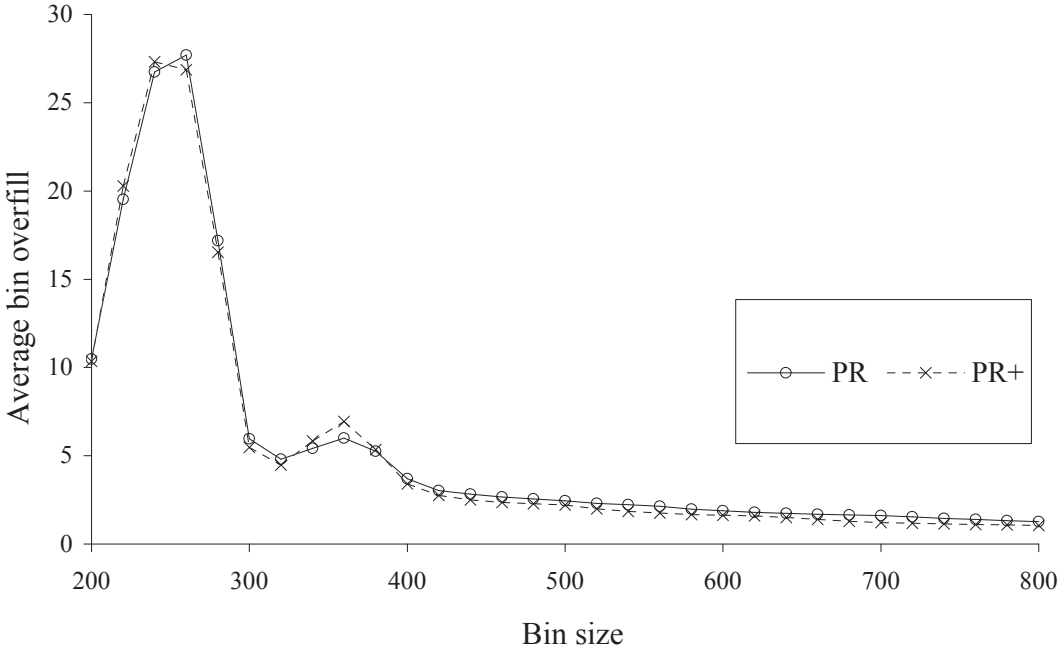


Figure 5.8: Comparing PR and PR+ algorithms for eight active bins, bin size from 200 to 800, and the $N_D(100,20)$ distribution.

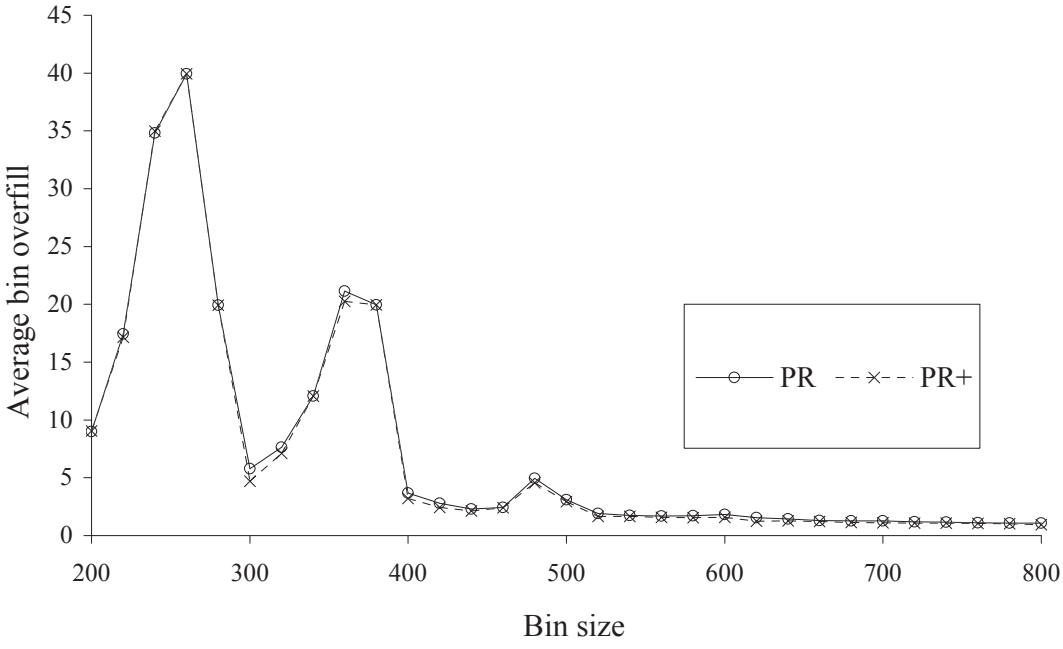


Figure 5.9: Comparing PR and PR+ algorithms for eight active bins, bin size from 200 to 800, and the $U_D(75,125)$ distribution.

An interesting effect can be seen as number of bins increases. The magnitude of this effect varies with distribution and bin size; to show it clearly a bin size of 600 and the $N_D(100,20)$ distribution are selected. The Optimal Zone Search Algorithm (3.3) is used, and each simulation is run for 60,000 bins with a 1000 bin warmup that is ignored (the simulation is shorter than before because we are not calculating confidence intervals). Figure 5.10 shows how the overweight drops for both algorithms as the number of bins increases, and that the overweight first decreases smoothly, but then stalls at approximately 0.5 for several adjacent numbers of active bins. The reason for this is that the optimal zone size decreases to 1, and stays there for a while until there are enough active bins to enable the optimal zone size to drop down to 0, as seen in Figure 5.11.

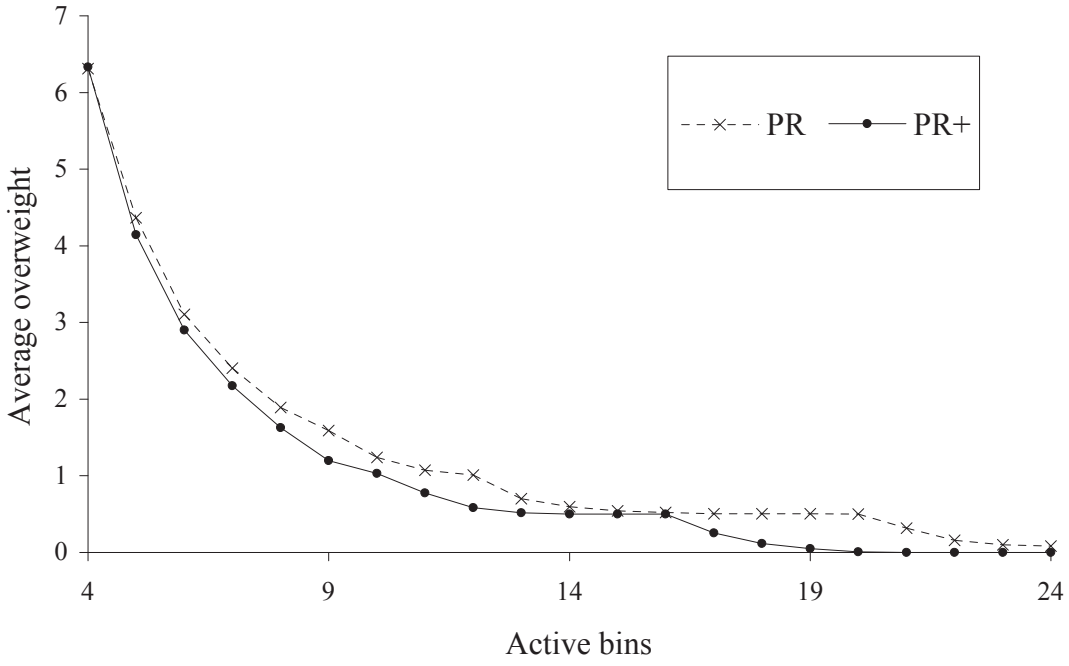


Figure 5.10: Comparison of the performance of the Prospect and Prospect+ Algorithms. Bin size is 600, the number of active bins is varied, and the $N_D(100,20)$ distribution is used.

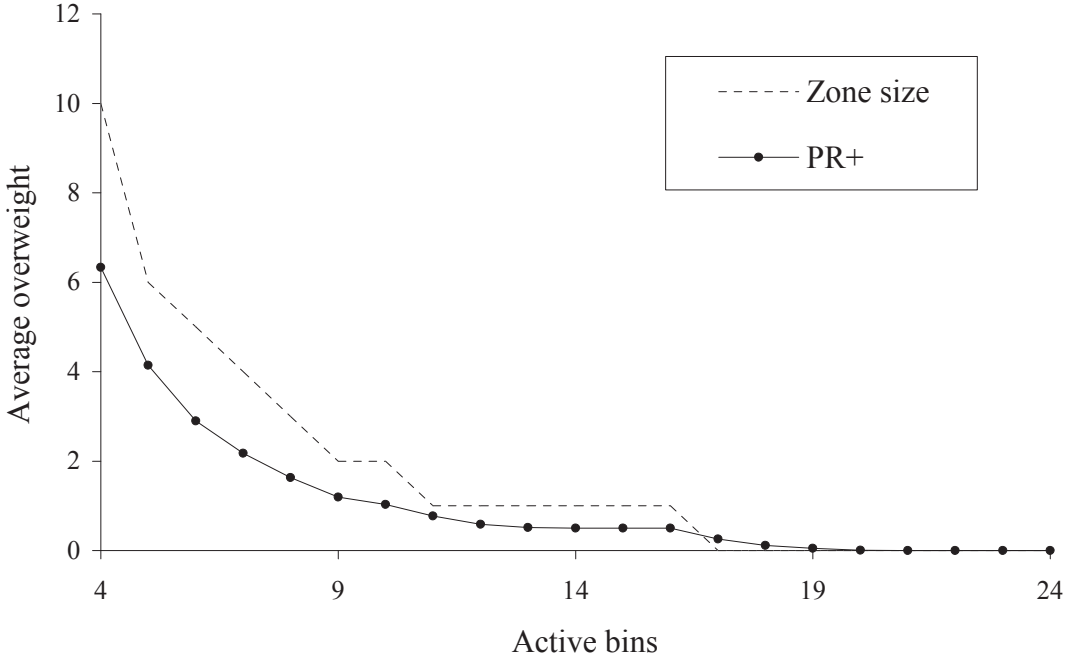


Figure 5.11: Comparison of the optimal zone size and average overweight for the Prospect+ Algorithm. Bin size is 600, the number of active bins is varied, and the $N_D(100,20)$ distribution is used.

CHAPTER 6

MARKOV MODELING OF ON-LINE BIN-COVERING

6.1 Introduction

If the items can only have a few different sizes and there is only a small number of active bins, then one can develop a *Markov* model of the Bin-Covering problem with a manageable number of states. This model can be used to determine the exact expected performance of the Prospect Algorithm, and it can be used in a *Markov Decision Process (MDP)* to determine a packing strategy that is, depending on the solution method, either optimal or ε-optimal. Being ε-optimal means that the performance of the solution is within a pre-described small number $\varepsilon > 0$ from the performance of the optimal solution.

The outline of this chapter is as follows. In Section 6.2, a state space model for Bin-Covering algorithms is defined. Then, the MDP identifying the optimal Bin-Covering strategy is described in Section 6.3, and its performance is compared to that of the Prospect Algorithm. In Section 6.4, a different MDP is introduced that models the true prospect for the Bin-Covering system (see Section 3.2). In Section 6.5, insights from the optimal MDP approaches of Sections 6.3 and 6.4 are used to improve on the Prospect Algorithm. In Section 6.6, the MDP approach is extended to solve practically sized problems, i.e., problems with a few hundred different item sizes and up to eight active bins. Finally in Section 6.7, a rollout version of the Prospect Algorithm that uses MDP ideas to achieve improved performance is tested. The Rollout Prospect Algorithm applies to practically sized problems, as long as the available processing power allows enough rollout simulations to be run to improve performance. The MDP analysis and modeling in this chapter is based mainly on

Puterman [97], but Bertsekas [13, 14] was used as a reference as well.

6.2 Bin State-Space Model

As in Chapter 3, it is assumed that both the item sizes and the bin size are integer-valued with no common divisors. The latter assumption is without loss of generality as it can be satisfied with an appropriate choice of the unit of measurement. Let b be the bin size, k be the number of active bins, and $f_{\max}$ and $f_{\min}$ be the largest and smallest possible item sizes, respectively. The state variable is a k-dimensional vector $\mathbf{s} \in \mathbb{N}^k$ that specifies the quantities in each of the k bins. It is assumed that only *possible* bin quantities are included, i.e., quantities that can be created by summing the sizes of a set of items. The entries in the vector $\mathbf{s}$ are sorted by size in ascending order, so that $s_1 \leq s_2 \leq \cdots \leq s_k$. A filled bin is modeled explicitly by allowing the last entry of $\mathbf{s}$, s_k, to become $\geq b$. The other entries are always less than b.

We will use integer labels to refer to the states. In this case, the state space is called $\mathcal{S}$, with the subclasses of states with all unfilled bins and a single filled bin labeled $\mathcal{S}_{ub}$ and $\mathcal{S}_{fb}$, respectively. If N_s is defined as the number of possible states, then $\mathcal{S} = \{1, 2, \ldots, N_s\}$. If we define N_{ub} and N_{fb} as the number of states with unfilled and filled bins, respectively, then $N_s = N_{ub} + N_{fb}$, $\mathcal{S}_{ub} = \{1, 2, \ldots, N_{ub}\}$, and $\mathcal{S}_{fb} = \{N_{ub} + 1, N_{ub} + 2, \ldots, N_s\}$.

This state space model was initially developed by the author in 2001, but E. Asgeirsson and C. Stein use the same state space model in their papers [3, 4], published in 2006 and 2009, respectively, as well as in E. Asgeirsson's PhD thesis [2] from 2007. They were unaware of the work done for this thesis, and developed their model independently. In [2, 3, 4], a formulation for the optimal solution is stated, but not solved. They use the Markov chain model to help them design a heuristic algorithm. The heuristic they employ introduces a penalty parameter that needs to be adjusted or optimized according to the item distribution, bin size, and number of active bins,

similarly as the zone parameter of the Prospect Algorithm has to be adjusted or optimized. The extra parameter is denoted α and they describe it on pages 465–466 of [4] as follows (B is the bin size in their notation):

> "We set the penalty for each open bin as $\max(0, h+\alpha-B)$, where h is the level in the bin and α is a number such that the probability of receiving an item no greater than α is small. ... The optimal value for α depends on the item distribution and the number of bins that we can have open."

In essence, they are working with a heuristic algorithm that utilizes a single parameter. In this sense, their heuristic resembles the Prospect Algorithm put forth in this thesis. By contrast we go all the way with the MDP formulation in this chapter; namely, we define the state space model and use it to both calculate the performance of the Prospect Algorithm, as well as obtain an optimal solution for the problem for comparison.

The following lemma can be used to calculate the number of possible bin states N_s.

Lemma 6.1 *Let $n_{ub} \leq b$ be the number of possible values for the quantity in an unfilled bin, and $n_{fb} \leq f_{\max}$ be the number of possible values for the quantity in a filled bin. Then the number of possible values for s_i is n_{ub} if $i \in \{1, 2, \ldots, k-1\}$ and $n_{ub} + n_{fb}$ if $i = k$. The number of possible states, N_s, is given by the following equation, where $\binom{n}{k}$ is the binomial coefficient:*

$$N_s = \binom{n_{ub}+k-1}{k} + n_{fb}\binom{n_{ub}+k-2}{k-1}. \tag{6.1}$$

Proof First count the states where there is no bin filled, N_{ub}. For this purpose, note that such states involve a sorted list of k entries, where each entry can have one of n_{ub} different values. This is the same as selecting k numbers in increasing order and with repetition out of the set of n_{ub} possible values, and the number of such

Table 6.1: Number of bin states as a function of n_{ub}, n_{fb}, and k.

Active bins k	n_{ub} / n_{fb}	6 / 7	20 / 10	50 / 20	100 / 60	200 / 100
1		13	30	70	160	300
2		63	410	2275	11,050	40,100
3		203	3,640	47,600	474,700	$\approx 3.4 \times 10^6$
4		518	24,255	734,825	$\approx 1.5 \times 10^7$	$\approx 2.0 \times 10^8$
5		1,134	131,054	$\approx 9.0 \times 10^6$	$\approx 3.6 \times 10^8$	$\approx 9.7 \times 10^9$
6		2,226	602,140	$\approx 9.2 \times 10^7$	$\approx 7.1 \times 10^9$	$\approx 3.8 \times 10^{11}$
7		4,026	$\approx 2.4 \times 10^6$	$\approx 8.1 \times 10^8$	$\approx 1.2 \times 10^{11}$	$\approx 1.2 \times 10^{13}$
8		6,831	$\approx 8.8 \times 10^6$	$\approx 6.3 \times 10^9$	$\approx 1.8 \times 10^{12}$	$\approx 3.6 \times 10^{14}$

possible combinations is given by $\binom{n_{ub}+k-1}{k}$ (see, e.g., formula (3.2.1) on Page 23 of Cameron [19]), which is the first entry in Equation (6.1). Now consider the states having one filled bin, N_{fb}. Note that in such states the first $k-1$ items form a sorted list as described before (just having one less item), and each such combination has n_{fb} possible different values of the kth item; so this yields $n_{fb}\binom{n_{ub}+k-2}{k-1}$ additional states, which is the second entry of Equation (6.1). ■

If the vector **s** were not sorted by size, then it is easy to see that the number of possible states would be equal to $n_{ub}^{k-1}(n_{ub}+kn_{fb})$, which is in general a much greater quantity than that given in Equation (6.1). In Table 6.1, the number of bin states for a few different values of n_{ub} and n_{fb} is shown to demonstrate how rapidly the state space grows.

A numerical example of the state mapping is given in Tables 6.2 and 6.3. The bin size is 20, there are two bins, and the items are from the distribution given in Table 2(a). Here $n_{ub} = 6$ and $n_{fb} = 7$, so Table 6.1 shows that there are 13 or 63 states if the number of active bins k is one or two, respectively. The state space for a single bin is shown in Table 6.2(b). Table 6.3 shows all the states when $k = 2$.

Given the bin states, one can model any given Bin-Covering algorithm as a Markov chain, provided that the item distribution is i.i.d. and the algorithm depends only on the current bin state, the weight of the next item, and the item distribution. Only

Table 6.2: Sample item distribution and single bin state space. Bin size is 20.

(a) Distribution

Item size	Probability
9	0.25
10	0.50
11	0.25

(b) Single bin state space

State no.	Size in bin
1	0
2	9
3	10
4	11
5	18
6	19
7	20
8	21
9	22
10	27
11	28
12	29
13	30

deterministic algorithms are of interest here, so the algorithms are assumed to be deterministic even though it is not required. To create a Bin-Covering Markov chain, two different events are modeled to obtain the transition matrix of the chain, namely, the new item event and the full bin event. For the new item event, which applies to states in $\mathcal{S}_{ub}$ (all bins unfilled), note that there is at most one weight $w_{i,j,A}$ that results in a transition from i to j under a Bin-Covering algorithm A. For the full bin event, which applies to states in $\mathcal{S}_{fb}$ (one bin filled), define the function $\mathrm{R}(i)$ which returns the state that results when the filled bin in state $i \in S_{fb}$ is set to zero and the state vector is resorted. Calculating the average and the distribution of the overfill is what is of interest here, and by having an extra full bin event transition, it is convenient to calculate these quantities. With $\mathcal{W}$ denoting the set of all possible item weights and $f(w)$ the probability of obtaining an item of size $w \in \mathcal{W}$, the nonzero entries of the transition matrix are filled by the following two equations:

$$p_{i,j} = f(w_{i,j,A}),\ \forall i \in \mathcal{S}_{ub} \text{ and } j \in \mathcal{S} \text{ for which } w_{i,j,A} \text{ exists}, \tag{6.2}$$

$$p_{i,\mathrm{R}(i)} = 1, \forall i \in \mathcal{S}_{fb}, \tag{6.3}$$

Table 6.3: Sample state space for two bins.

State no.	Bin 1	Bin 2	State no.	Bin 1	Bin 2
1	0	0	33	10	27
2	0	9	34	10	28
3	0	10	35	10	29
4	0	11	36	10	30
5	0	18	37	11	11
6	0	19	38	11	18
7	0	20	39	11	19
8	0	21	40	11	20
9	0	22	41	11	21
10	0	27	42	11	22
11	0	28	43	11	27
12	0	29	44	11	28
13	0	30	45	11	29
14	9	9	46	11	30
15	9	10	47	18	18
16	9	11	48	18	19
17	9	18	49	18	20
18	9	19	50	18	21
19	9	20	51	18	22
20	9	21	52	18	27
21	9	22	53	18	28
22	9	27	54	18	29
23	9	28	55	18	30
24	9	29	56	19	19
25	9	30	57	19	20
26	10	10	58	19	21
27	10	11	59	19	22
28	10	18	60	19	27
29	10	19	61	19	28
30	10	20	62	19	29
31	10	21	63	19	30
32	10	22			

where Equation (6.2) corresponds to the new item event, and Equation (6.3) corresponds to the full bin event.

To illustrate, the state space given in Table 6.3 is used. If $i = 16$ ($\mathbf{s} = [9, 11]$) and $w = 10$, then j can either be 39 if the first bin is chosen, resulting in state $\mathbf{s} = [11, 19]$; or $j = 20$ if the second bin is chosen, resulting in state $\mathbf{s} = [9, 21]$. Assume that A chooses the second bin for the item, so that $w_{16,20,A} = 10$; then $p_{16,20} = f(10) = 0.5$ (see Table 2(a)). The resulting state $j = 20$ has a full bin that is emptied with $\mathrm{R}(j) = 2$ ($\mathbf{s} = [0, 9]$), so $p_{20,2} = 1$. Note that the resulting Markov chain will in most cases have both transient and recurrent states. Which states are recurrent and which are transient depends on the algorithm being modeled.

To analyze the performance of a given algorithm with a single recurrent class of states, and maybe some transient states as well, the steady-state distribution $\boldsymbol{\pi}$ of the appropriate Markov chain can be calculated. To calculate $\boldsymbol{\pi}$, Gauss–Seidel (G–S) iteration was used with the initial state $\boldsymbol{\pi}_0 = [1, 1, \ldots, 1]/N_s$. If the G–S iteration converges, it converges to the steady-state distribution [110]. If the whole state space was used, the G–S iteration did not always converge, probably because there were too many transient states. However, after a simple pruning involving elimination of all states that were never entered into (i.e., all states i with $\sum_{j \in \mathcal{S}} p_{ji} = 0$), the G–S iteration always converged in our calculations. We also experimented with using the *Fox–Landi* algorithm [59] (see also Puterman [97]) to eliminate any leftover transient states. Running the Fox–Landi algorithm usually took more time than what was saved in the subsequent G–S iteration, so we did not use it in our calculation runs.

Assuming a known steady-state distribution $\boldsymbol{\pi}$, the average overfill for algorithm A can be computed. The individual entries of $\boldsymbol{\pi}$, namely π_i, where $i \in \mathcal{S}$, represent the long-run proportion of time that the Markov chain is in each state i. For all $i \in S_{fb}$, let o_i be the overfill in the full bin state i. The average overfill $\bar{o}$ can be

calculated by a weighted average of the overfill in the full bin states:

$$\bar{o} = \frac{\sum_{i \in \mathcal{S}_{fb}} \pi_i o_i}{\sum_{i \in \mathcal{S}_{fb}} \pi_i}. \tag{6.4}$$

By analyzing the steady-state distribution of the full bin states, it is also easy to obtain the distribution of overfill. Define $\mathcal{O}$ as the set of all possible bin overfills; usually $\mathcal{O} = \{0, 1, \ldots, f_{\max} - 1\}$. Now the frequency of each possible overfill, $f'(\psi)$, is calculated by:

$$f'(\psi) = \frac{\sum_{i \in \mathcal{S}_{fb} \text{ with } o_i = \psi} \pi_i}{\sum_{i \in \mathcal{S}_{fb}} \pi_i}, \forall \psi \in \mathcal{O}. \tag{6.5}$$

6.3 Markov Decision Process Model

The objective of the Markov Decision Process model for Bin-Covering is to minimize the long–run average overfill of filled bins. Puterman [97] classifies this as an *Infinite Horizon* problem with the *Average Reward* criterion. The *reward* in this case is the bin overfill, and the goal is to minimize it. The MDP is discrete time, with decisions taken at discrete points in time called *decision epochs*, and the goal is to determine the optimal decision *action* at each decision epoch.

In our case, the decision epochs are when items arrive to be placed in bins. Thus the full-bin event of Equation (6.3) is not a decision epoch. The bin state space model in Section 6.2 is used, with the size of the next item, w, added, or $\tilde{\mathbf{s}} = [\mathbf{s}, w] = [s_1, s_2, \ldots, s_k, w]$. When in state $\tilde{\mathbf{s}}$ at a decision epoch t, the action a is the index of the bin chosen to receive the item w, i.e., $a \in \{1, 2, \ldots, k\}$. The reward $r_t(\tilde{\mathbf{s}}, a)$ at decision epoch t is:

$$r_t(\tilde{\mathbf{s}}, a) = \max(0, s_a + w - b). \tag{6.6}$$

Since items arrive at decision epochs to be placed in bins, the transition probability at a decision epoch t is given by the item distribution $f(w)$:

$$p_t(\tilde{\mathbf{s}}_{w',a} | \tilde{\mathbf{s}}, a) = f(w'), \tag{6.7}$$

where $\tilde{\mathbf{s}}_{w',a}$ is the resulting system state after item w is put into bin a, bin a is emptied if needed, the bin entries reordered, and a new item w' generated.

Issues regarding MDP modeling that are important for our bin-covering problem are discussed below. For a more in-depth description of MDP modeling, the reader is referred to Puterman [97]. Note that in this work, an MDP refers to an MDP as it applies to our problem, i.e., an Infinite Horizon MDP with the Average Reward Criterion.

Puterman [97] classifies an MDP as *communicating* if for every pair of states s and $\hat{s}$ in the state space $\mathcal{S}$, there exists a deterministic stationary policy d^∞ under which $\hat{s}$ is accessible from s. Moreover, the MDP is *weakly communicating* if there exists a *closed* subset of communicating states under some deterministic stationary policy, plus a possibly empty set of states which is transient under every policy; and *multichain* if the transition matrix corresponding to *at least one* stationary policy contains two or more closed irreducible recurrent classes. Theorem 6.2 states that our MDP problem is communicating and can be multichain.

Theorem 6.2 *The MDP resulting from the state space model described in Equations (6.2) and (6.3) of Section 6.2 can be* multichain *and is* communicating.

Proof This proof is split into two parts. The first part is an example that shows that the transition matrix corresponding to a stationary policy can contain two closed irreducible recurrent classes, showing that the MDP is classified as multichain. The second part argues that the MDP problem is communicating.

Part 1, multichain example. Assume that the distribution is made out of items of sizes 4 and 5 only, and the bin size is 10. The fraction of items of each item size does not matter. The optimal strategy, provided that there are two or more bins available, is simply to put 4's in one bin and 5's in another, because then all filled bins will be of size 12 (three 4's) or 10 (two 5's). If 4's and 5's are mixed in a bin, the resulting size of a filled bin will be higher, namely 13 $(4 + 4 + 5)$ or

14 $(4 + 5 + 5)$, which is obviously suboptimal. This means that if there are five or more bins, an optimal algorithm with multichain structure is easy to devise by controlling how it uses the extra "unnecessary" bin(s). For example, if there are five bins, then an example of a multichain and optimal policy is one that will never mix items of different sizes in a bin, puts the initial three items in three empty bins, and after that always puts an item in an empty bin, provided that an empty bin is available and either there is no bin with the same size item already, or there are two or three bins already with the same size item. If there are no available empty bins under the above criteria, the algorithm will put the item in the fullest bin with items of the same size (ties can be resolved arbitrarily). For example, if the first three items have more 4's than 5's, this algorithm will always use one bin for 5's and four for 4's, and vice versa if the first three items have more 5's than 4's. The fact that the policy in the example is also an optimal policy is not required, but it does make the example more interesting.

Part 2, the MDP is communicating. A state $\tilde{\mathbf{s}}$ is a vector of k bin state quantities plus the next item. There is at least one set of items from the distribution that sums up to each given bin state quantity. Therefore, a randomized stationary policy that simply assigns each new item to a bin randomly will eventually hit all the states of the MDP and as the state space is finite, the underlying Markov chain is recurrent. Then, by part a of Proposition 8.3.1 of Puterman [97], the MDP is communicating. ■

The result of the classification means that either *policy iteration* or *value iteration* can be used to solve the MDP, as stated in Sections 9.5.1 and 9.5.3 of Puterman [97]. Value iteration can be used as long as the optimal gain of the MDP is constant for all states, which is the case if the MDP is communicating, and the transition matrices are aperiodic. If the transition matrices are periodic, then they can be transformed to aperiodic matrices, but that is not necessary in our MDP's. Policy iteration can

solve communicating MDP's, but a more-complex version is needed if the MDP is multichain because then the gain of a stationary policy is possibly nonconstant. We choose value iteration, mainly because it is simpler to program and hence quicker to develop. Policy iteration was kept as a backup if problems with value iteration were encountered, but that was not needed as the program was fast enough for our purpose.

Since we are using an infinite horizon and average reward formulation in our MDP, we choose to use *relative value iteration* as defined in Section 8.5.5 of Puterman [97]. In relative value iteration a single reference state is chosen, and after each iteration the cost-to-go estimate for the reference state is used to normalize the cost-to-go values of all the states. For the reference state, the relative value iteration gives an estimate of the optimal gain, which in this case is the average overfill incurred at a decision epoch, averaged over all decision epochs. The cost-to-go values for all states will be the relative cost compared to the reference state.

Given an estimate of the relative cost-to-go after $N-1$ iterations, $V_{N-1}(\tilde{\mathbf{s}})$, then after N iterations, the estimate of the optimal gain g_N is given by the following equation, with the state with all bins empty and $w = f_{\min}$ selected as the reference state (i.e., $\tilde{\mathbf{s}}^* = [\mathbf{0}, f_{\min}]$):

$$g_N = \min_a \left(r(\tilde{\mathbf{s}}^*, a) + \sum_{w'=f_{\min}}^{f_{\max}} f(w') V_{N-1}(\tilde{\mathbf{s}}^*_{w',a}) \right). \tag{6.8}$$

For all states $\tilde{\mathbf{s}}$, the relative value function equals

$$V_N(\tilde{\mathbf{s}}) = \min_a \left(r(\tilde{\mathbf{s}}, a) + \sum_{w'=f_{\min}}^{f_{\max}} f(w') V_{N-1}(\tilde{\mathbf{s}}_{w',a}) \right) - g_N. \tag{6.9}$$

In our MDP model, the decision epochs correspond to adding new items, so the gain is the average overfill incurred when adding a new item. The gain itself is not what is of interest here, but rather the average overfill $\bar{o}$ of the optimal algorithm. Because the MDP is communicating, it follows from Theorems 8.3.2 and 9.1.8 in

Puterman [97] that the optimal gain is constant across all states. Moreover, it is possible to calculate the average overfill $\bar{o}$ directly from the gain g using Equations (6.10) and (6.11) from Propositions 6.3 and 6.4 (see Equation (6.12) below), implying that $\bar{o}$ is also constant across all states.

Proposition 6.3 *Multiplying the average overfill per item, g, with the average number of items per bin, $\bar{i}$, yields the average overfill per bin:*

$$\bar{o} = \bar{i}g. \tag{6.10}$$

Proof Define, after n items have been processed, M_n as the number of filled bins, $\bar{i}_n = n/M_n$ as the average number of items per filled bin, $\bar{y}_n$ as the average filled bin size, $\bar{o}_n = \bar{y}_n - b$ as the average bin overfill, and g_n as the average gain. Also define Y_j as the weight of filled bin $j \in \{1, 2, \ldots, M_n\}$. Now:

$$\begin{aligned}
\bar{o}_n &= \bar{y}_n - b \\
&= \frac{1}{M_n} \sum_{j=1}^{M_n} (Y_j - b) \\
&= \frac{n}{M_n} \times \frac{1}{n} \sum_{j=1}^{M_n} (Y_j - b) \\
&= \bar{i}_n g_n.
\end{aligned}$$

Applying the limit $n \to \infty$ to the above gives, with probability one, that

$$\bar{o} = \lim_{n\to\infty} \bar{o}_n = \lim_{n\to\infty} (\bar{i}_n g_n) = \bar{i}g.$$

■

Proposition 6.4 *Bin overfill can be written as a function of the average item size, μ, items per bin, $\bar{i}$, and bin size b:*

$$\mu\bar{i} - b = \bar{o}. \tag{6.11}$$

Proof Define, after n items have been processed, μ_n as the average item size and U_n as the sum of item weights in unfilled bins. Also define X_i as the weight of item $i \in \{1, 2, \ldots, n\}$. Note that U_n is bounded above by $(b-1)k$. Now:

$$\begin{aligned}
\mu_n &= \frac{1}{n}\sum_{i=1}^{n} X_i \\
&= \frac{1}{n}\left(\sum_{j=1}^{M_n} Y_j + U_n\right) \\
&= \frac{M_n}{n} \times \frac{1}{M_n}\sum_{j=1}^{M_n} Y_j + \frac{U_n}{n} \\
&= \frac{\bar{o}_n + b}{\bar{i}_n} + \frac{U_n}{n};
\end{aligned}$$

see the proof of Proposition 6.3. Applying the limit $n \to \infty$ to the above gives, with probability one, that

$$\mu = \lim_{n\to\infty} \mu_n = \lim_{n\to\infty}\left(\frac{\bar{o}_n + b}{\bar{i}_n} + \frac{U_n}{n}\right) = \frac{\bar{o} + b}{\bar{i}},$$

which yields Proposition 6.4 after a simple reorganization of the variables. ■

By eliminating $\bar{i}$ from Propositions 6.3 and 6.4, and solving for $\bar{o}$, we get the following relationship:

$$\frac{gb}{\mu - g} = \bar{o}. \tag{6.12}$$

As long as $b > 0$, it is safe to divide by $\mu - g$ since the average gain will always be less than the average item size (see Equation (6.6)). Equation (6.12) shows that minimizing the average gain also minimizes the average overfill, and, moreover, that the average overfill can be calculated directly from the gain, instead of using the steady-state distribution of the Markov chain corresponding to the optimal strategy to calculate the average overweight.

Once that was working, the slowest steps (creation of the states of the Markov model and the actual value iteration) were rewritten in C and linked to the Matlab code as MEX-functions. For the analysis, scaled-down versions of some of the

distributions from Chapter 3 are used, namely, three discrete distributions with rational probabilities that mimic the normal (Gaussian) distributions N_D(10,1.0), N_D(10,1.5), N_D(10,2.0) and a discrete uniform distribution $U_D(8, 12)$ (which has $(\mu, \sigma) = (10, 1.58)$). The relative variance (σ/μ) of these distributions is similar to the relative variance of the distributions used in the preceding chapters, which makes comparisons with other parts of this thesis more meaningful. These distributions are provided in Appendices A.6, A.7, A.8, and A.10, respectively. The model was run on a 2 Ghz Pentium Dual Core laptop with 2 GB of internal memory. On this machine, it was practical to consider problems with three active bins up to a bin size of 80, so the calculations were done for all bin sizes from 20 to 80 in increments of 2. When the problems grow in size, the calculation time increases because of both the larger number of states and the slower iteration convergence of the cost-to-go function. The initial cost-to-go vector was set to all zeros and the value iteration was run until the maximum error was below 10^{-12}.

When running the value iterations for the four distributions, two convergence issues were encountered. Both issues occurred for highly constrained cases, where there was little scope for any algorithm to reduce the overfill. For example, in the case when the bin size is 26 and item sizes range from 8 through 12 (such as in the discrete N_D(10,1.0) and U_D(8,12) distributions), all bins get filled with exactly 3 items unless one puts either three 8's for a total of 24 or two 8's and a 9 for a total of 25 in a bin, causing a fourth item to be needed to fill the bin. In this case the optimization basically becomes that of minimizing the number of bins that get filled with 4 items. The value iteration convergence can be very slow in these heavily constrained cases, often because the cost-to-go vector starts to oscillate between two values and hence converges extremely slowly. To fight this issue, two modes of cost-to-go vector update were used. Initially the cost-to-go vector is set to the newest available value at the end of each iteration, but if slow convergence is detected, the cost-to-go vector is set

to the average value of the last two cost-to-go vectors. The final iteration is, however, always done with a full update. We denote the two types of updates as *full update* and *half update*, respectively. The decision to switch between the two cost-to-go update methods was determined as follows:

- Define $10^0, 10^{-1}, 10^{-2}, \ldots, 10^{-12}$ as iteration goals. When the value iteration maximum error goes below an iteration goal, it is said to *hit an iteration goal*.
- When in full update, switch to half update if the next iteration goal is not hit in 5,000 iterations.
- When in half update, switch to full update if the next iteration goal is not hit in 100 iterations.

Incorporating half updates is equivalent to re-initializing the value iteration, and hence it is a valid modification to the value iteration approach. Using the above method to switch between half and full updates, it was possible to make all but one scenario converge. The toughest case was that of the N_D(10,1.0) distribution with bin size 26. In this case, both iteration types started to converge extremely slowly once the maximum error got to approximately 3.31×10^{-8}. To get around that issue, the initial cost-to-go vector was set to be the final cost-to-go vector for the U_D(8,12) distribution, utilizing the fact that both distributions have the same range of items (8 through 12) and hence an identical state space. This cost-to-go vector converged in 5,001 extra iterations. Table 6.4 shows the number of value iterations needed to obtain the solution within the prescribed error. Bold entries indicate where half update iterations were employed, and the * indicates the special case mentioned above.

Figures 6.1 to 6.4 show the average overfill of the optimal strategy compared with the average overfill of the Prospect Algorithm from Chapter 3 for the item distributions N_D(10,1.0), N_D(10,1.5), N_D(10,2.0), and $U_D(8, 12)$, respectively. The

Table 6.4: Number of value iterations needed to get maximum error below 10^{-12}. Boldfaced entries indicate that half updates were employed.

	Distribution			
Bin size	$N_D(10,1.0)$	$N_D(10,1.5)$	$N_D(10,2.0)$	$U_D(8,12)$
20	333	274	214	343
22	163	140	153	176
24	1009	238	188	338
26	**48087+5001***	856	334	**48087**
28	**12940**	4157	850	**7074**
30	1038	755	564	785
32	299	448	352	499
34	509	473	383	565
36	3746	834	493	1254
38	**13071**	1494	632	**6288**
40	2382	1394	670	1966
42	1068	978	633	1131
44	1632	1015	678	1176
46	2593	1170	747	1364
48	8749	1505	828	1675
50	5286	1602	899	3738
52	2669	1557	951	1754
54	2411	1616	1008	1833
56	2624	1756	1075	1961
58	7571	1905	1150	2115
60	5900	2042	1230	2307
62	4707	2149	1304	2475
64	3968	2242	1378	2522
66	6242	2360	1452	2664
68	6587	2496	1532	2815
70	6947	2649	1617	2984
72	6516	2786	1702	3150
74	6512	2902	1786	3284
76	7419	3028	1869	3437
78	7590	3174	1955	3594
80	8170	3330	2045	3763

average overfill of the Prospect Algorithm is calculated exactly using the Markov model described in Section 6.2. The PR model is run over all possible zone sizes, which are 0 to $f_{\max} - 1$, and then the best one is selected and shown in Figures 6.1 to 6.4. As an example, we illustrate this process in Figure 6.5 for bin sizes 32, 42, and 62 and the $N_D(10,1.5)$ distribution.

All overfill curves in Figures 6.1 to 6.4 show the same basic pattern of decreasing peaks and valleys:

- When the bin size is close to being an even multiple of the average item size, the overfill drops, especially if the bin size is small.
- As the bin size increases, the overfill drops.

The optimal strategy often has significantly lower overfill than the Prospect Algorithm. In Figure 6.6, the average overfill graph for the Prospect Algorithm, using the $N_D(10,1.5)$ distribution, is extended with simulation up to bin size 200 (each simulation is run until 10,000 bins have been filled). These simulations demonstrate that the overfill of the Prospect Algorithm keeps decreasing towards zero, clearly indicating that the overfill gap between the Prospect Algorithm and the optimum (but unknown) solution continues to decrease as the bin size increases.

Tables 6.5 and 6.6 list the average overfill for the MDP and Prospect Algorithms and the ratio of the overfills for the four distributions. The optimal MDP algorithm generates between 7.0% and 100.0% of the overfill that the Prospect Algorithm with the optimal zone size generates in the runs. The difference in performance of the two algorithms is zero (100% ratio) for the most-constrained cases where any algorithm will be forced to give significant overfill.

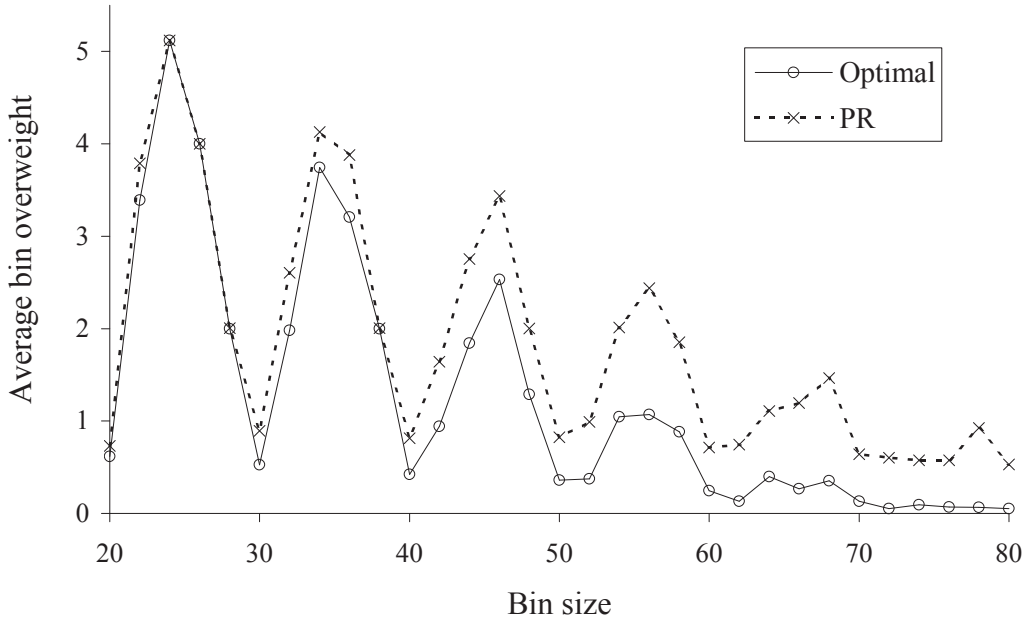


Figure 6.1: Effects of varying bin size on the Optimal and PR algorithms for three active bins and the $N_D(10,1.0)$ distribution.

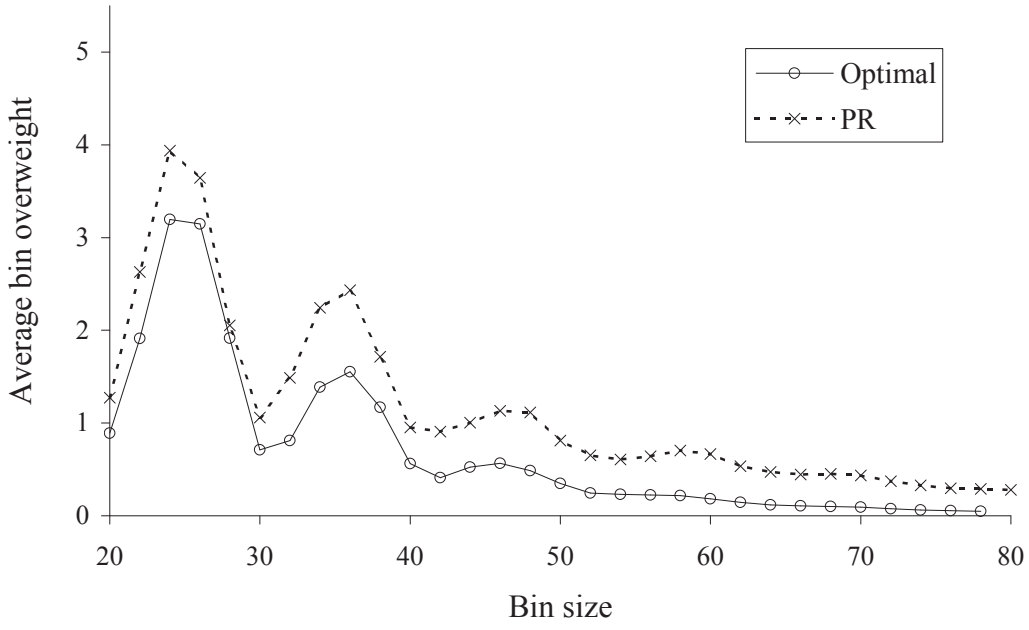


Figure 6.2: Effects of varying bin size on the Optimal and PR algorithms for three active bins and the $N_D(10,1.5)$ distribution.

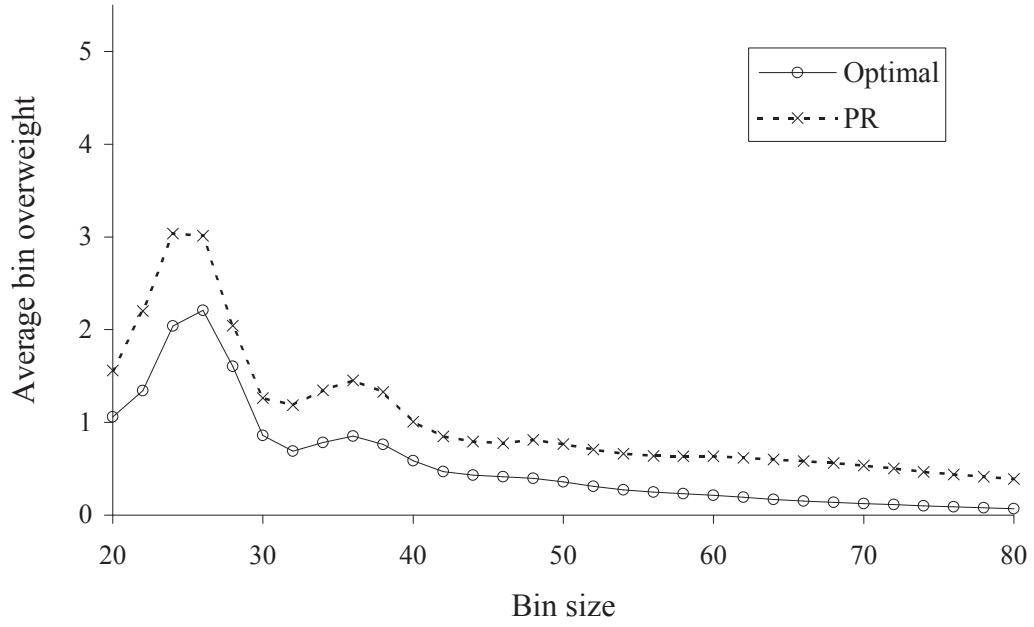


Figure 6.3: Effects of varying bin size on the Optimal and PR algorithms for three active bins and the $N_D(10,2.0)$ distribution.

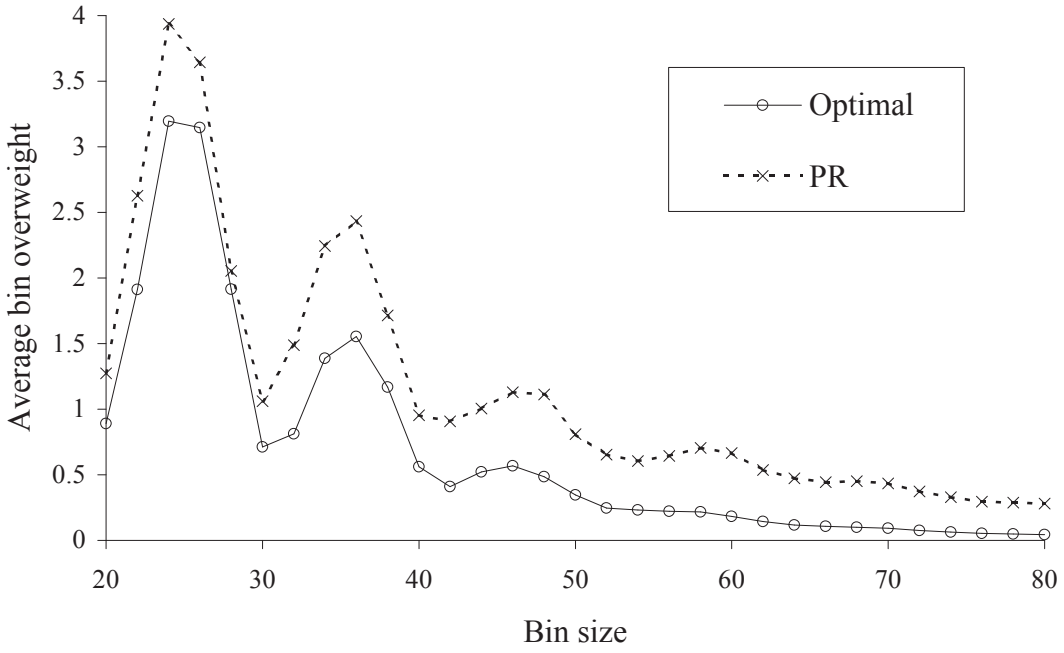


Figure 6.4: Effects of varying bin size on the Optimal and PR algorithms for three active bins and the $U_D(8,12)$ distribution.

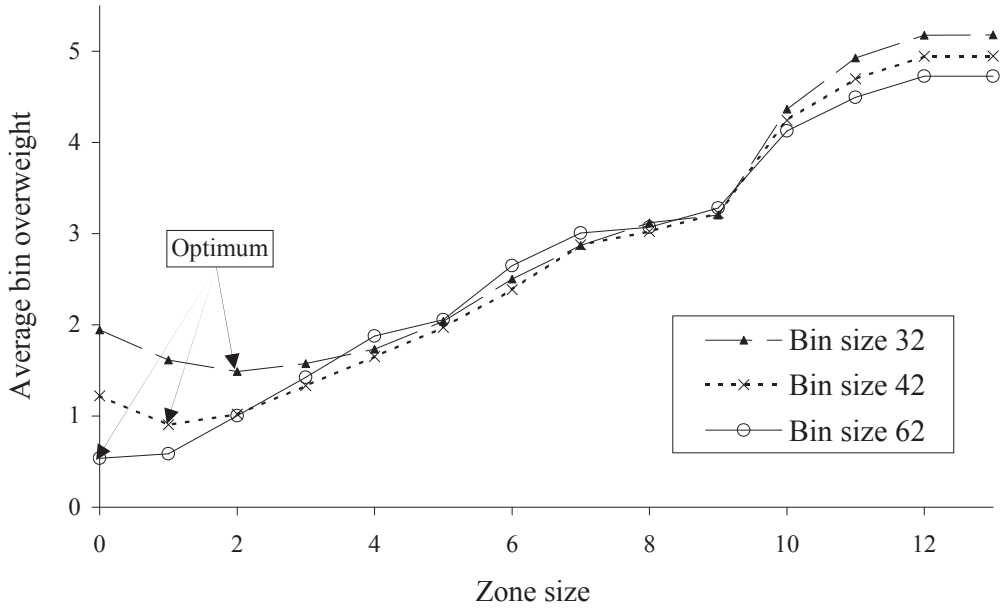


Figure 6.5: Effect of zone size for three active bins of sizes 32, 42, and 52 and the $N_D(10,1.5)$ distribution.

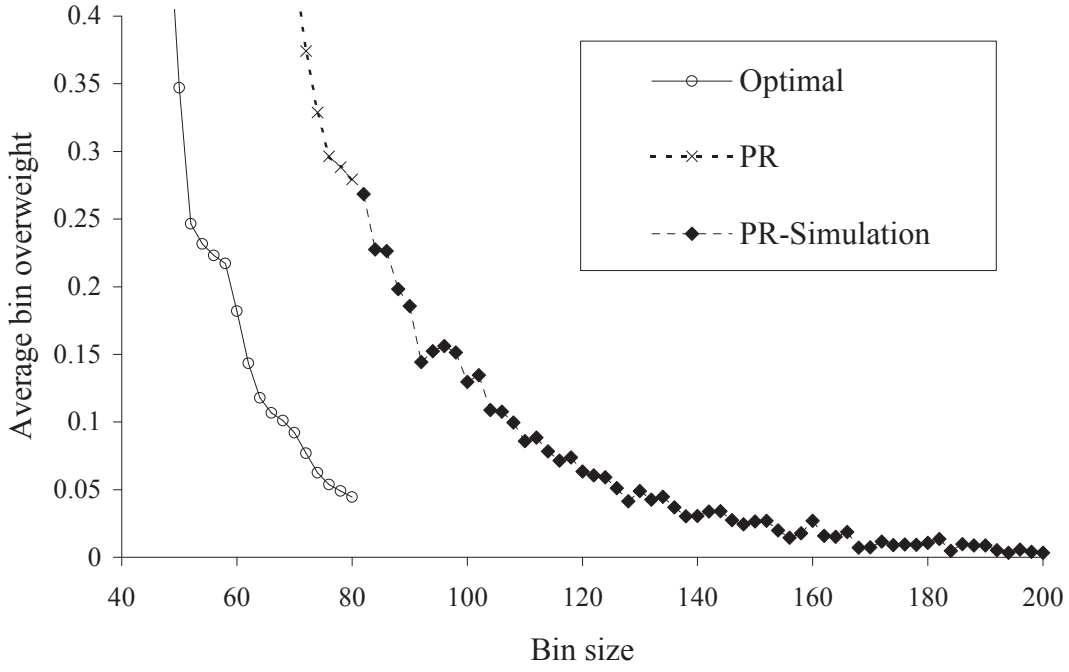


Figure 6.6: Performance of the PR Algorithm for three active and large bins and the $N_D(10,1.5)$ distribution.

Table 6.5: Overfill comparison I of the MDP and Prospect Algorithms ($N_D(10,1.0)$ and $N_D(10,1.5)$ distributions).

	$N_D(10,1.0)$			$N_D(10,1.5)$		
Bin size	MDP	PR	MDP/PR	MDP	PR	MDP/PR
20	0.6179	0.7301	84.6%	0.8891	1.274	69.8%
22	3.392	3.790	89.5%	1.912	2.628	72.8%
24	5.118	5.118	100.0%	3.194	3.936	81.2%
26	4.000	4.000	100.0%	3.146	3.644	86.3%
28	2.000	2.004	99.8%	1.916	2.054	93.3%
30	0.5280	0.8944	59.0%	0.7113	1.060	67.1%
32	1.982	2.604	76.1%	0.8117	1.488	54.6%
34	3.745	4.130	90.7%	1.387	2.243	61.8%
36	3.208	3.879	82.7%	1.553	2.434	63.8%
38	2.000	2.004	99.8%	1.169	1.715	68.2%
40	0.4208	0.8124	51.8%	0.5619	0.9528	59.0%
42	0.9409	1.643	57.3%	0.4104	0.9065	45.3%
44	1.842	2.754	66.9%	0.5230	1.005	52.0%
46	2.530	3.435	73.7%	0.5675	1.130	50.2%
48	1.288	2.003	64.3%	0.4857	1.113	43.6%
50	0.3598	0.8238	43.7%	0.3471	0.8107	42.8%
52	0.3745	0.9891	37.9%	0.2465	0.6536	37.7%
54	1.047	2.013	52.0%	0.2315	0.6060	38.2%
56	1.069	2.438	43.9%	0.2231	0.6429	34.7%
58	0.8839	1.853	47.7%	0.2172	0.7054	30.8%
60	0.2456	0.7142	34.4%	0.1819	0.6658	27.3%
62	0.1309	0.7459	17.6%	0.1433	0.5357	26.8%
64	0.3993	1.109	36.0%	0.1177	0.4725	24.9%
66	0.2682	1.194	22.5%	0.1068	0.4442	24.0%
68	0.3532	1.464	24.1%	0.1009	0.4504	22.4%
70	0.1302	0.6398	20.4%	0.09208	0.4348	21.2%
72	0.05330	0.6028	8.84%	0.07680	0.3742	20.5%
74	0.09337	0.5740	16.3%	0.06240	0.3287	19.0%
76	0.07022	0.5753	12.2%	0.05366	0.2963	18.1%
78	0.06466	0.9262	6.98%	0.04893	0.2883	17.0%
80	0.05146	0.5300	9.71%	0.04451	0.2795	15.9%

Table 6.6: Overfill comparison II of the MDP and Prospect Algorithms ($N_D(10,2.0)$ and $U_D(8,12)$ distributions).

	$N_D(10,2.0)$			$U_D(8,12)$		
Bin size	MDP	PR	MDP/PR	MDP	PR	MDP/PR
20	1.061	1.560	68.0%	0.8309	1.160	71.7%
22	1.343	2.202	61.0%	1.627	2.075	78.4%
24	2.041	3.038	67.2%	3.273	3.273	100.0%
26	2.208	3.012	73.3%	4.000	4.000	100.0%
28	1.604	2.044	78.5%	2.004	2.034	98.5%
30	0.8583	1.262	68.0%	0.7231	1.086	66.6%
32	0.6894	1.187	58.1%	0.6713	1.216	55.2%
34	0.7847	1.347	58.2%	0.9973	1.560	64.0%
36	0.8521	1.451	58.7%	1.519	2.363	64.3%
38	0.7638	1.329	57.5%	2.008	2.047	98.1%
40	0.5880	1.007	58.4%	0.5263	0.9577	55.0%
42	0.4703	0.8499	55.3%	0.3224	0.7877	40.9%
44	0.4317	0.7927	54.5%	0.3365	0.6977	48.2%
46	0.4134	0.7760	53.3%	0.3865	0.9037	42.8%
48	0.3965	0.8095	49.0%	0.3909	1.432	27.3%
50	0.3572	0.7653	46.7%	0.4887	0.8833	55.3%
52	0.3112	0.7077	44.0%	0.1821	0.6680	27.3%
54	0.2741	0.6642	41.3%	0.1540	0.6010	25.6%
56	0.2494	0.6397	39.0%	0.1494	0.6031	24.8%
58	0.2318	0.6336	36.6%	0.1492	0.7307	20.4%
60	0.2140	0.6338	33.8%	0.1469	0.7342	20.0%
62	0.1925	0.6181	31.1%	0.1206	0.5488	22.0%
64	0.1699	0.5994	28.3%	0.08419	0.4869	17.3%
66	0.1511	0.5848	25.8%	0.07383	0.4457	16.6%
68	0.1367	0.5634	24.3%	0.06763	0.4967	13.6%
70	0.1247	0.5336	23.4%	0.06536	0.5023	13.0%
72	0.1126	0.5026	22.4%	0.06050	0.4443	13.6%
74	0.09986	0.4674	21.4%	0.04751	0.3853	12.3%
76	0.08810	0.4382	20.1%	0.03784	0.3475	10.9%
78	0.07831	0.4127	19.0%	0.03278	0.3683	8.90%
80	0.07027	0.3915	17.9%	0.03079	0.3608	8.54%

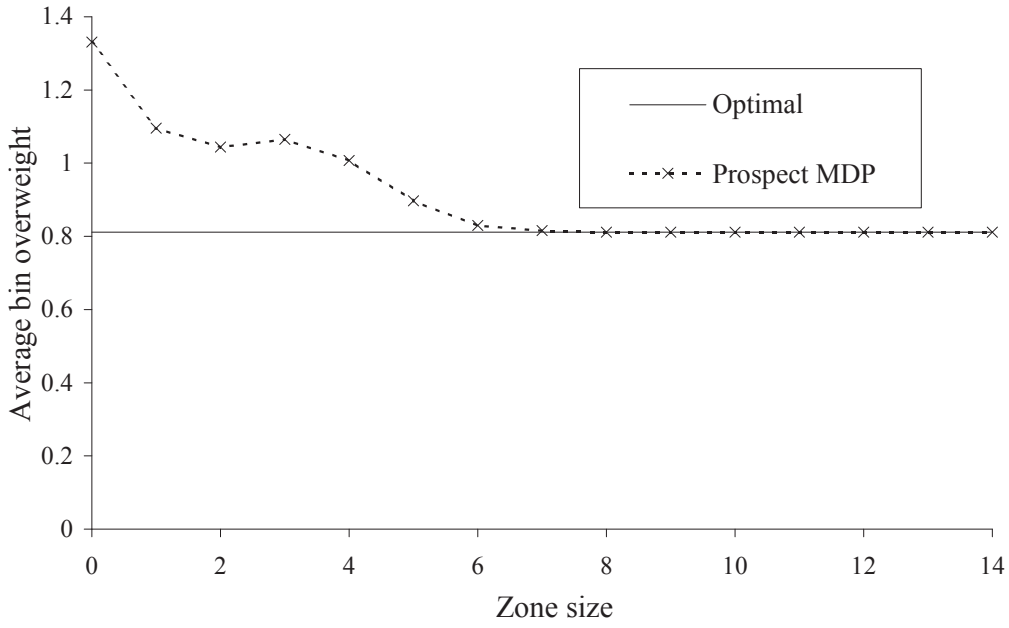


Figure 6.7: Effects of varying zone size for the Prospect MDP with three active bins, bin size 32, and the $N_D(10,1.5)$ distribution.

6.4 Markov Decision Process Model for the Prospect Function

It is simple to modify the MDP of the previous section so that it optimizes the Prospect function of the Bin-Covering system (see Page 24). The problem has to be changed to a maximization problem with the objective as an indicator function that is one if the bin is within the defined zone, and zero otherwise. The required reward function is:

$$r_t(\tilde{\mathbf{s}}, a) = \mathrm{I}_{\{b \le s_a + w \le b + z\}}. \tag{6.13}$$

The model is called the Prospect MDP, and Figure 6.7 shows the results of that model for the $N_D(10,1.5)$ distribution and varying zone size, compared to the optimal MDP.

It may seem surprising that the performance of the Prospect MDP becomes optimal as the zone size increases. But this is expected behavior because if the zone parameter is high enough (greater than or equal to the largest item weight minus 1), then it has no effect, and the Prospect MDP model maximizes the number of bins

generated. This is obviously the same as minimizing the average overfill (see also Equation (6.11)), and hence optimal. In the example in Figure 6.7, the maximum item size is 14 (see Appendix A.7), so it is expected that the Prospect MDP does not equal the optimal MDP until the zone size is 13 (one minus the biggest item since that is the maximum filled bin). The fact that the optimum is reached before that (for zone size 9) is caused by the bounds on the numeric accuracy in the computer modeling — the fraction of bins that get filled at weight above 32+9 is similar to or smaller than the maximum error in the optimal MDP solution (10^{-12}). The non–monotonicity of the overfill with regard to the zone size (i.e., the hump observed for zone = 3) probably reflects the fact that the prospect rewards (Equation (6.13)) do not account explicitly for overfill, but indirectly via the number of bins generated within the desired zone size.

6.5 Improving the Prospect Algorithm Using Insights from the Optimal Strategy

When comparing the optimal Bin-Covering strategy and the Prospect Algorithm, a significant difference was discovered, and this led to a modification to the Prospect Algorithm. To demonstrate this, Figures 6.8 and 6.9 compare the overfill distributions for the Optimal MDP algorithm and the Prospect Algorithm (with optimal zone size) for the $N_D(10,1.5)$ item distribution, plotted on linear and logarithmic scales, respectively. The other distributions showed similar results. The graph with the logarithmic scale is included because it shows more accurately the details of the overfill distribution as the frequency drops close to zero. The distributions are significantly different. In those figures, the bin size is 40 and the optimal zone size is 1 (optimal over all possible zone sizes 0 through 13), so the Prospect Algorithm is trying to finish batches with 0 or 1 overfill, making those two finishing positions the most likely. If a piece has to be forced into a bin, the resulting overfill may be significantly higher than the zone size. The result is that the majority of bins finish with 0 or

1 overfill, then there is a steep drop in the frequency of bins with overfill 2, and after that the frequency increases again, peaks at 5 and then starts to drop off again. The optimal strategy, on the other hand, has an overfill distribution with strictly decreasing frequency as the overfill increases.

The idea is to modify the Prospect Algorithm so that it generates batches with overfill distribution that resembles the optimal strategy, and that can be done by modifying how the zone, z, is defined. In the Prospect Algorithm, z basically defines a square-shaped evaluation curve, that evaluates a bin with overfill less than or equal to z as 1, and 0 otherwise. This evaluation curve yields the overfill distribution that is observed in Figures 6.8 and 6.9, suggesting that if the shape of the evaluation curve is changed so that it resembles the optimal overfill distribution, the actual overfill distribution might follow a similar shape. With this in mind a new, smooth evaluation curve to evaluate the bin overfill is proposed, namely evaluate a bin with zero overfill as 1, and then discount bins with more overfill with a discount factor. Define the Prospect Ratio Exponential (PRE) algorithm in exactly the same way as the Prospect Algorithm except that instead of using the prospect function of Equation (3.2), it uses the following function to measure the prospect:

$$\mathrm{Pfe}(w,r) = \begin{cases} 0 & \text{if } w \leq -f_{\max}, \\ r^{-w} & \text{if } -f_{\max} < w \leq 0, \\ \sum_{i=0}^{f_{\max}-1} r^i \mathrm{Pf}(w+i) & \text{if } w > 0. \end{cases} \tag{6.14}$$

In this formulation, a discount factor, r, has to be set instead of the zone. The other parameter w is the weight left in the bin, and the function $\mathrm{Pf}(w)$ is the same as defined in Section 3.2. Note that in the case that $r = 0$, we define $0^0 = 1$.

In the following performance comparisons, the optimal value for r is selected after looping through all values from 0.00 through 1.00 in 0.05 increments. Note that setting $r = 1.0$ is the same as setting zone as $z = f_{\max} - 1$ and using the Prospect

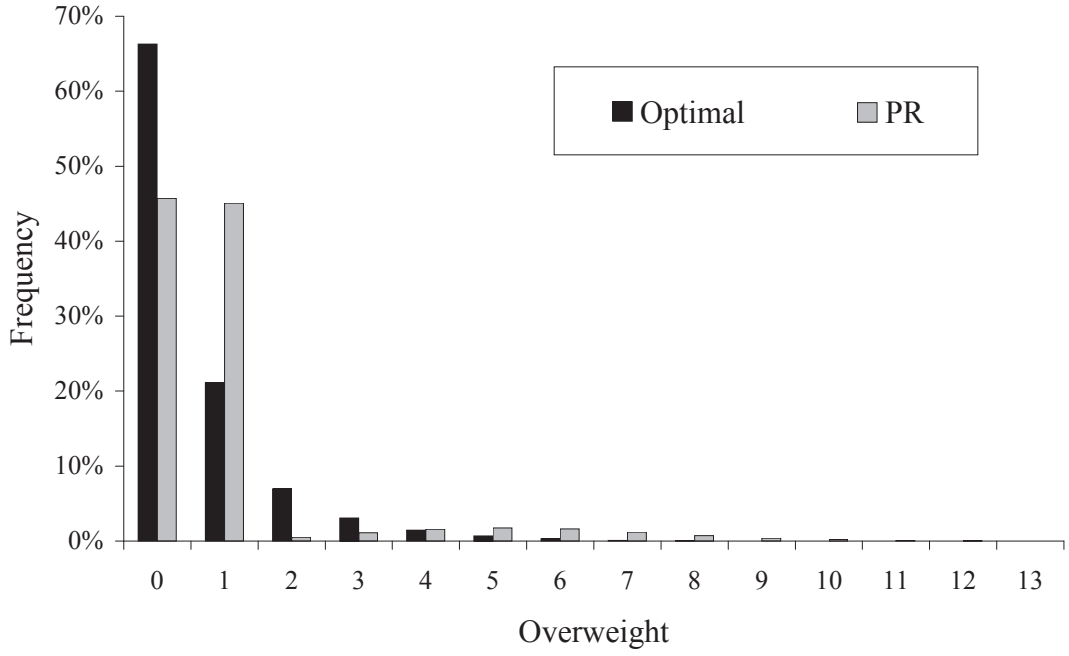


Figure 6.8: Comparison of overfill distributions for the Optimal and Prospect Algorithms with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

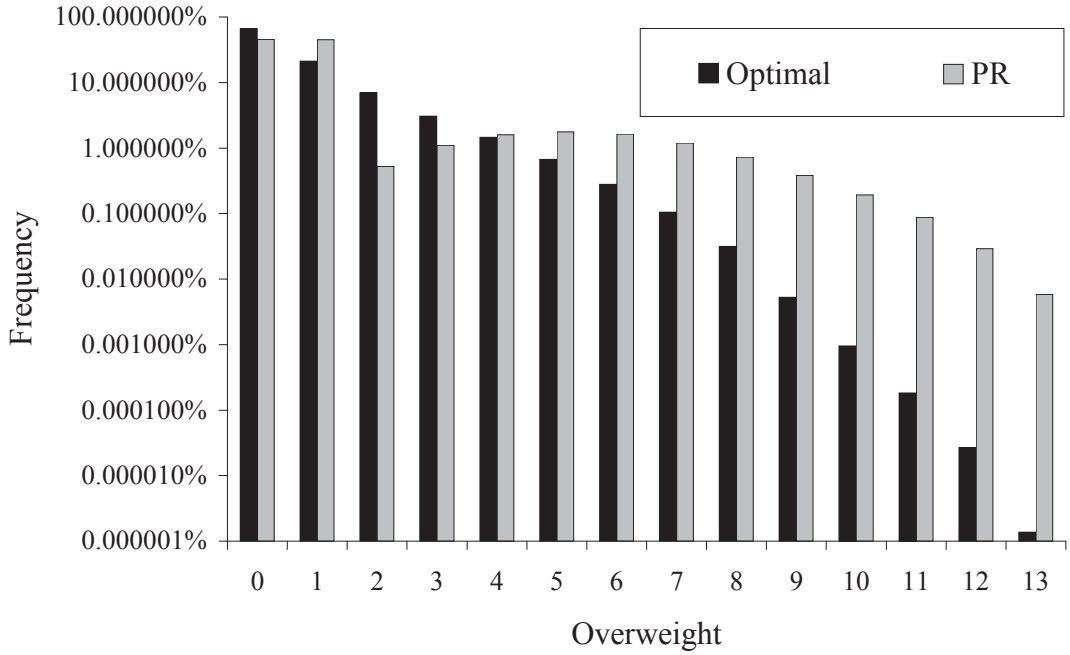


Figure 6.9: Comparison of overfill distributions for the Optimal and Prospect Algorithms on a logarithmic scale with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

Algorithm. Figure 6.10 compares the two types of evaluation curves with the optimal value for r (i.e., $r = 0.15$) when there are three active bins, the bin size is 40, and $N_D(10,1.5)$ is the item distribution. In Figures 6.11 through 6.14, the overfill of the PRE algorithm has been added to Figures 6.1 through 6.4. These results clearly show that in this case, PRE outperforms PR significantly. In Figures 6.15 and 6.16, the overfill distribution for PRE has been added to Figures 6.8 and 6.9. This shows that the overfill distribution looks more like the distribution of the optimal strategy.

Figure 6.17 compares the parameter sensitivity of the PR and PRE algorithms. The PR zone varies from 0 to 13 but is scaled onto a [0,1] range to enable plotting on the same scale as the PRE algorithm. Figure 6.17 shows that PRE is less sensitive to its parameter r around the optimal value than the Prospect Algorithm. In Table 6.7, the optimal values for r in Figures 6.11 through 6.14 are listed. Apart from the bin sizes where the achievable overfill is high, the optimal value for r does not vary a great deal and is usually within 0.15 and 0.45 (75% of the values are within that range). This, combined with the fact that PRE is usually not very sensitive to r around the optimal value, means that it is much easier to choose a value for r. It is more likely in practical instances that r can simply be preset to a single value and not changed at all. In Table 6.8, optimal values for the parameter r are shown, assuming that r is kept constant for all bin sizes. The optimal single constant r for all simulation runs is 0.20, and in Figures 6.18 through 6.21 the comparison of using PRE with optimal r and constant $r = 0.20$ is shown. Using constant r works well, especially when either the average overfill is low or the variance of the item size is not very small. When the average overfill is high and the item variance is small, it is better to use a slightly higher value for r.

In Figure 6.22, the PRE algorithm is compared via simulation to the Prospect Algorithm for a realistically sized problem. The bin size is 600, items are drawn from the $N_D(100,15)$ distribution (described in Appendix A.3), the number of bins

varied from 3 to 20, and in each simulation 1000 bins are filled for warmup and then 60,000 bins are filled to estimate the average overfill for each run. The optimal values for r are estimated as described above, and the optimal zone z is estimated with Algorithm 3.3. PRE outperforms PR if the number of bins is below 18, but when the number of bins is 18 or more the two algorithms are identical. The reason for that can be seen in Table 6.9 where the values for the optimal parameters z and r are listed. Both parameters start high and then decrease as the number of bins increases. The optimal zone size z is constant (equal to one) when the number of bins is between 11 and 17, and the optimal discount factor r is low but oscillates slightly in that range. Both parameters, however, drop to zero when the number of bins gets to 18 and above, and in that case both algorithms become identical since both aim at filling all bins with zero overfill. Also note that the relative margin that PRE beats PR by is smaller than in Figure 6.12. For example, when the number of bins in Figure 6.22 is three, the overfill of PR is only about 23% higher than that of PRE (9.88 vs 8.04), but is about 44% higher in Figure 6.12 when the bin size is 60 (0.666 vs 0.461). The reason is probably that when the granularity of the distribution increases, the relative advantage of the smooth zone of the PRE algorithm decreases. The distribution $N_D(10,1.5)$ used in Figure 6.12 has only 9 different sizes of items (6 through 14), whereas the distribution $N_D(100,15)$ used in Figure 6.22 has 91 different item sizes (55 through 145). Chapter 8 includes further comparisons of PR and PRE for realistically sized problems.

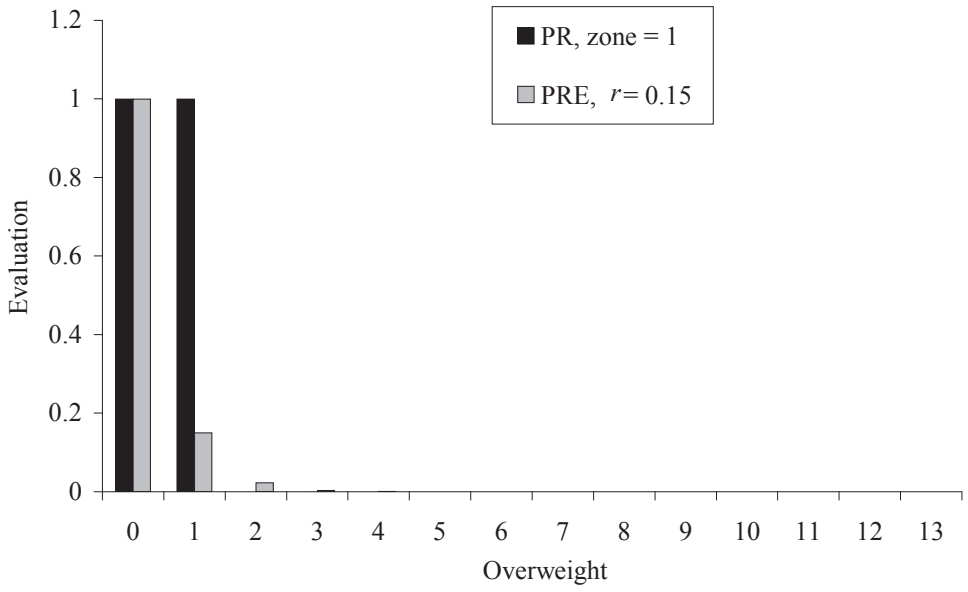


Figure 6.10: Comparison of square zone ($z = 1$) evaluation and optimal exponential decay ($r = 0.15$) evaluation with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

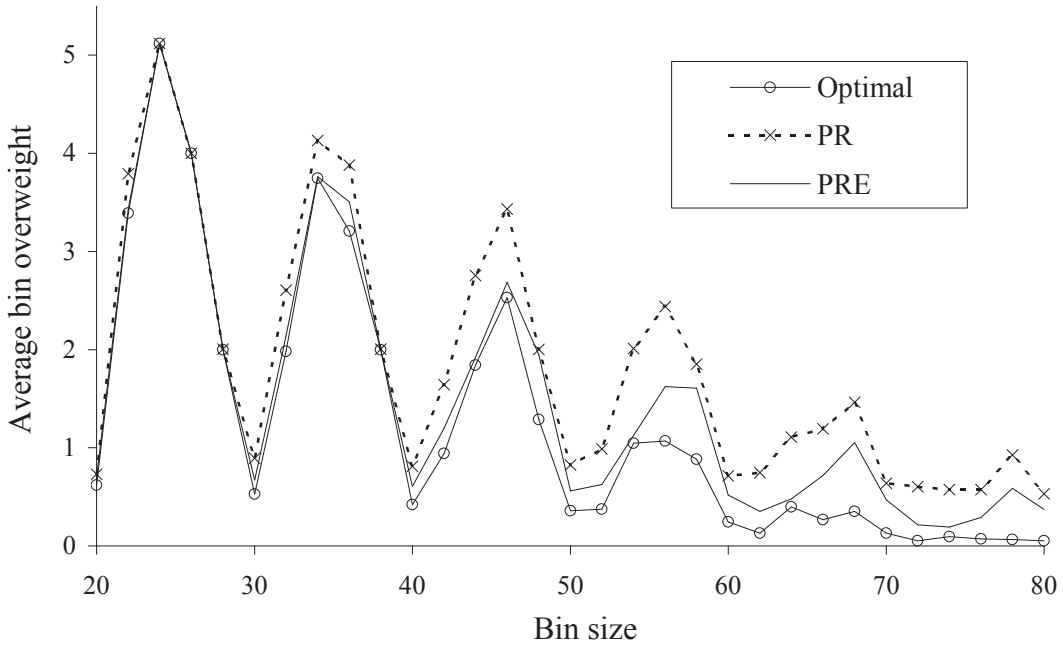


Figure 6.11: Effects of varying bin size on the Optimal, PR, and PRE algorithms for three active bins and the $N_D(10,1.0)$ distribution.

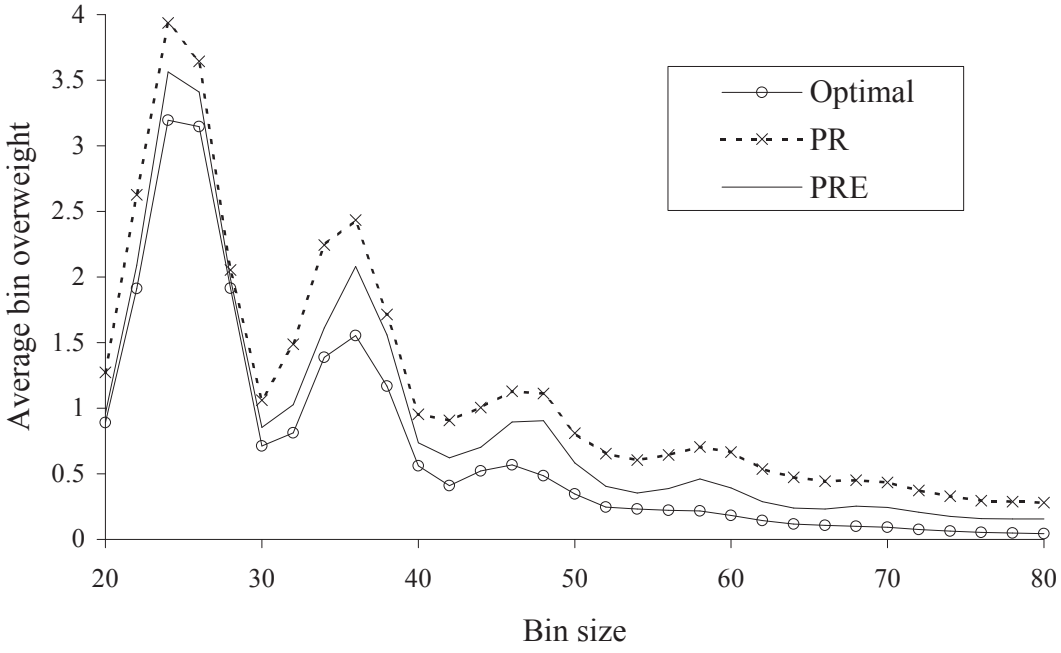


Figure 6.12: Effects of varying bin size on the Optimal, PR, and PRE algorithms for three active bins and the $N_D(10,1.5)$ distribution.

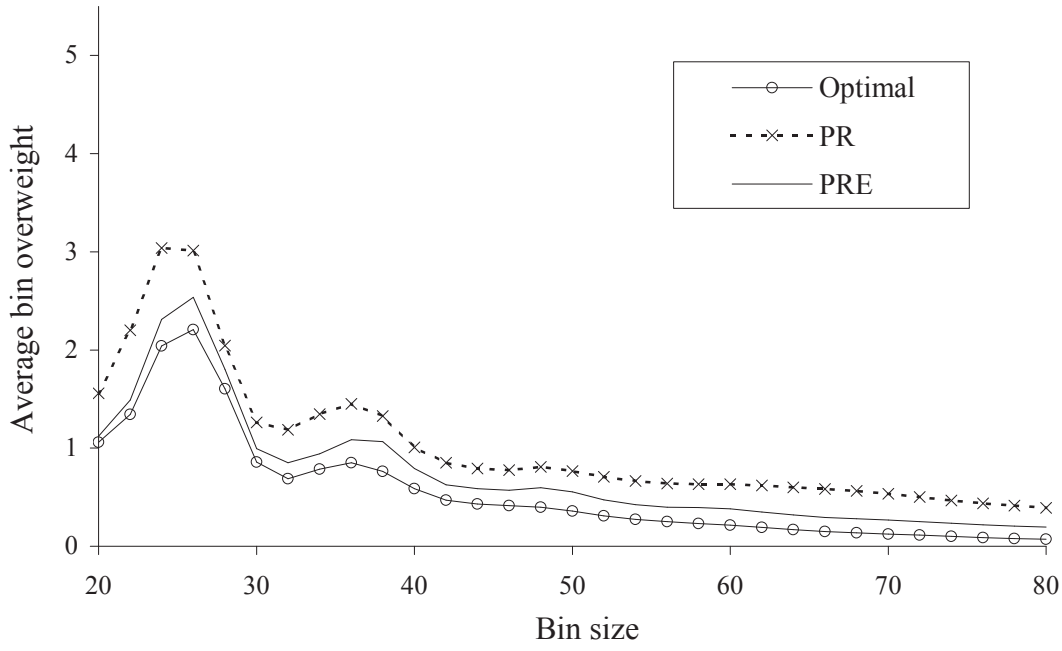


Figure 6.13: Effects of varying bin size on the Optimal, PR, and PRE algorithms for three active bins and the $N_D(10,2.0)$ distribution.

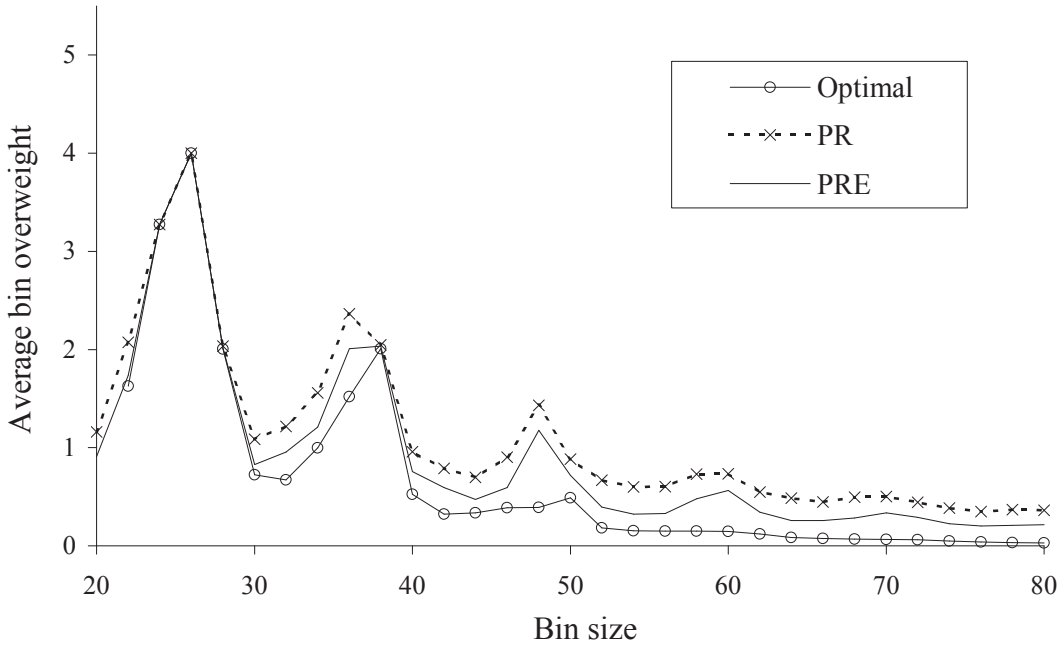


Figure 6.14: Effects of varying bin size on the Optimal, PR, and PRE algorithms for three active bins and the $U_D(8,12)$ distribution.

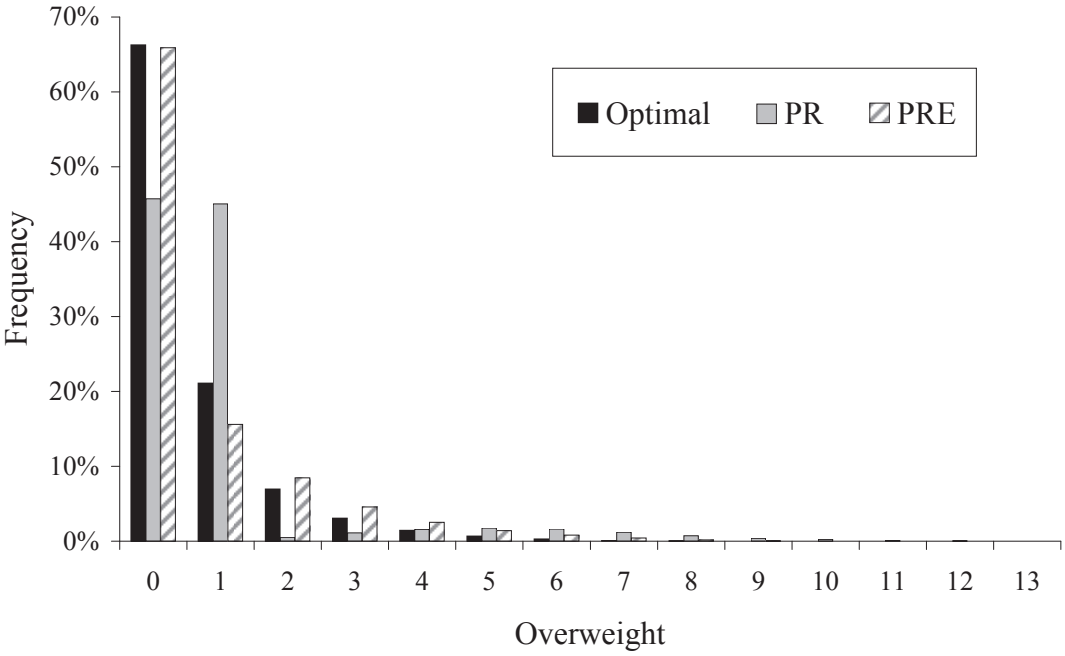


Figure 6.15: Comparison of overfill distributions for the optimal, PR, and PRE algorithms with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

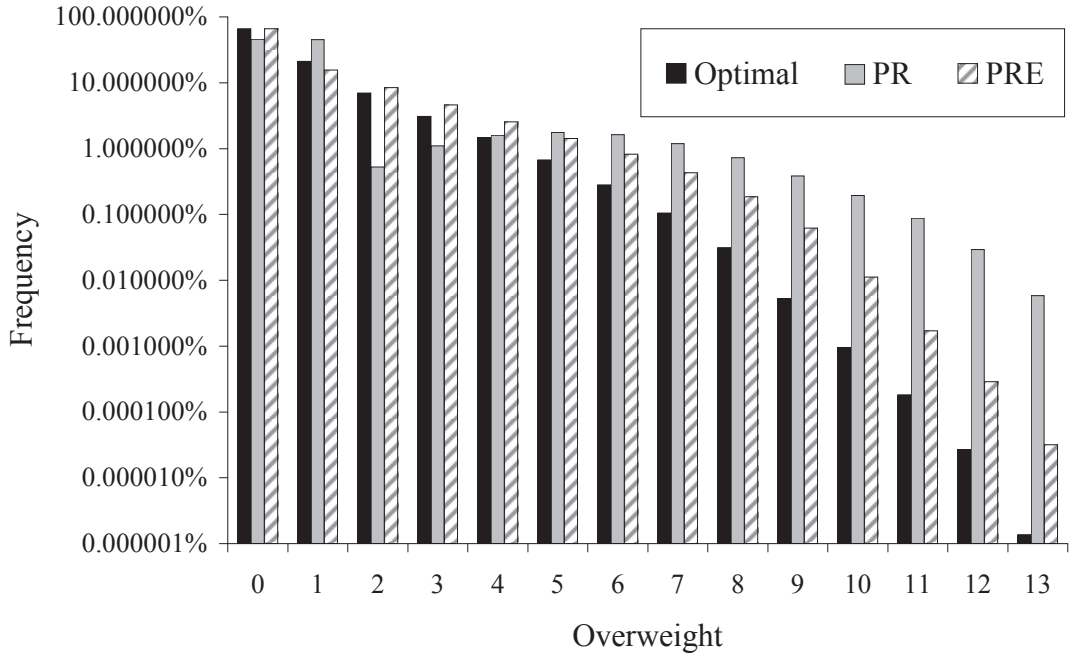


Figure 6.16: Comparison of overfill distributions for the optimal, PR, and PRE algorithms on a logarithmic scale with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

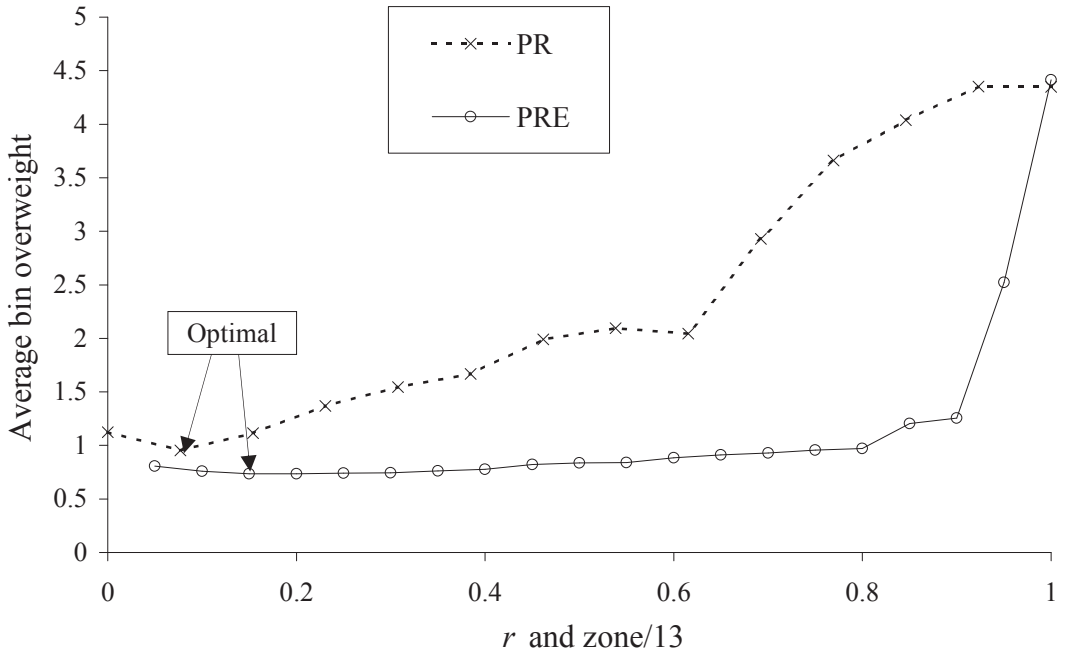


Figure 6.17: Comparison of parameter sensitivity between the PR and PRE algorithms with three active bins, bin size 40, and the $N_D(10,1.5)$ distribution.

Table 6.7: Optimal values for the parameter r of the PRE algorithm.

Bin size	$N_D(10,1.0)$	$N_D(10,1.5)$	$N_D(10,2.0)$	$U_D(8,12)$
20	0.55	0.20	0.35	0.15
22	0.90	0.85	0.40	0.60
24	0.15	0.70	0.65	0.50
26	0.45	0.35	0.60	0.35
28	0.05	0.15	0.45	0.35
30	0.70	0.25	0.30	0.20
32	0.75	0.55	0.40	0.90
34	0.45	0.65	0.45	0.50
36	0.40	0.50	0.45	0.50
38	0.10	0.40	0.35	0.25
40	0.55	0.15	0.25	0.20
42	0.55	0.30	0.25	0.25
44	0.50	0.35	0.20	0.20
46	0.25	0.40	0.20	0.15
48	0.05	0.35	0.20	0.10
50	0.55	0.20	0.25	0.05
52	0.60	0.30	0.20	0.15
54	0.45	0.25	0.20	0.15
56	0.40	0.20	0.20	0.15
58	0.15	0.25	0.15	0.15
60	0.45	0.20	0.20	0.05
62	0.45	0.20	0.15	0.15
64	0.40	0.20	0.20	0.20
66	0.35	0.15	0.15	0.10
68	0.30	0.15	0.15	0.15
70	0.45	0.15	0.15	0.10
72	0.45	0.15	0.15	0.15
74	0.35	0.15	0.15	0.15
76	0.20	0.15	0.15	0.05
78	0.20	0.15	0.15	0.15
80	0.15	0.15	0.15	0.15

Table 6.8: Average overfill for all bin sizes using constant values for the parameter r of the PRE algorithm. Optimal values are **bold**.

r	$N_D(10,1.0)$	$N_D(10,1.5)$	$N_D(10,2.0)$	$U_D(8,12)$	Average
0.05	1.8045	1.0723	1.0083	1.0522	1.2343
0.10	1.7861	1.0252	0.9064	1.0431	1.1902
0.15	1.7458	0.9779	0.8387	**0.9166**	1.1198
0.20	1.7100	0.9557	**0.7698**	0.9190	**1.0886**
0.25	1.6889	**0.9405**	0.8005	1.0665	1.1241
0.30	1.6696	0.9486	0.8481	1.1124	1.1447
0.35	1.5818	0.9804	0.8621	1.0428	1.1168
0.40	1.5499	1.0199	0.8676	1.0581	1.1239
0.45	**1.5480**	1.0478	0.8709	1.1904	1.1643
0.50	1.5632	1.0606	0.8793	1.1921	1.1738
0.55	1.5802	1.0742	0.9053	1.0989	1.1647
0.60	1.6113	1.1024	0.9317	1.1105	1.1890
0.65	1.6229	1.1471	0.9911	1.2444	1.2514
0.70	1.6627	1.1639	1.0565	1.2654	1.2871
0.75	1.7473	1.2005	1.0896	1.1867	1.3060
0.80	1.9063	1.2521	1.0801	1.2159	1.3636
0.85	2.1866	1.4540	1.0840	1.4217	1.5366
0.90	2.5854	2.1330	1.6470	1.9184	2.0709
0.95	3.6620	3.5212	3.6894	3.1007	3.4933
1.00	4.5591	4.6334	4.8969	4.5964	4.6715

Table 6.9: Optimal parameters for PR and PRE algorithms when bin size is 600, item data is $N_D(100,15)$, and the number of bins is varied.

Bins	z (PR)	r (PRE)	Bins	z (PR)	r (PRE)
3	12	0.80	12	1	0.10
4	7	0.70	13	1	0.05
5	6	0.65	14	1	0.10
6	4	0.50	15	1	0.05
7	3	0.35	16	1	0.10
8	3	0.30	17	1	0.10
9	2	0.25	18	0	0.00
10	2	0.15	19	0	0.00
11	1	0.15	20	0	0.00

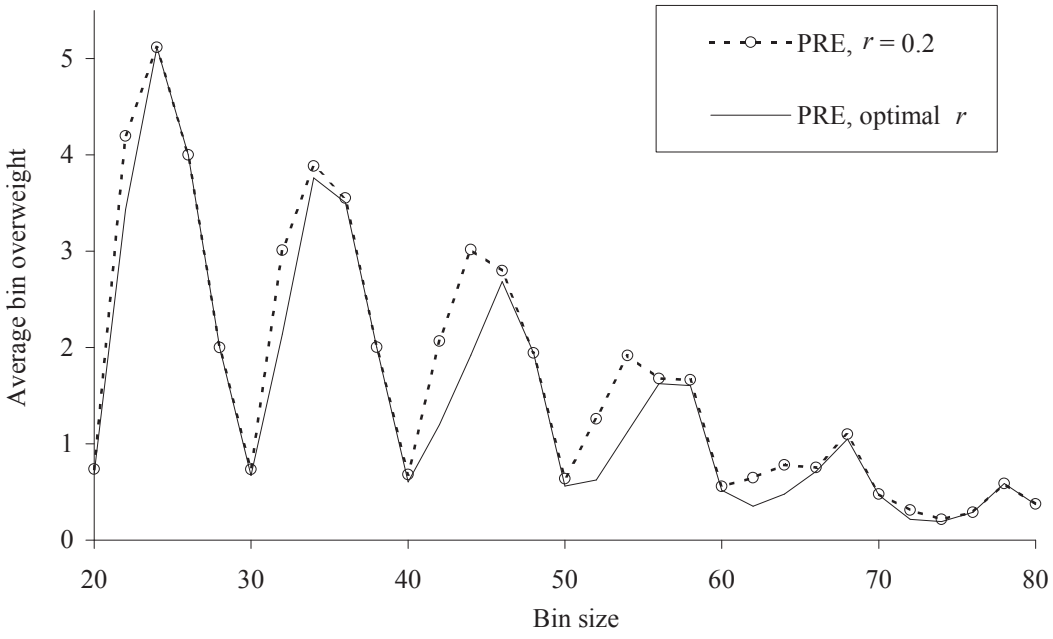


Figure 6.18: Effects of varying bin size on the PRE algorithm for three active bins and the $N_D(10,1.0)$ distribution when using constant $r = 0.20$.

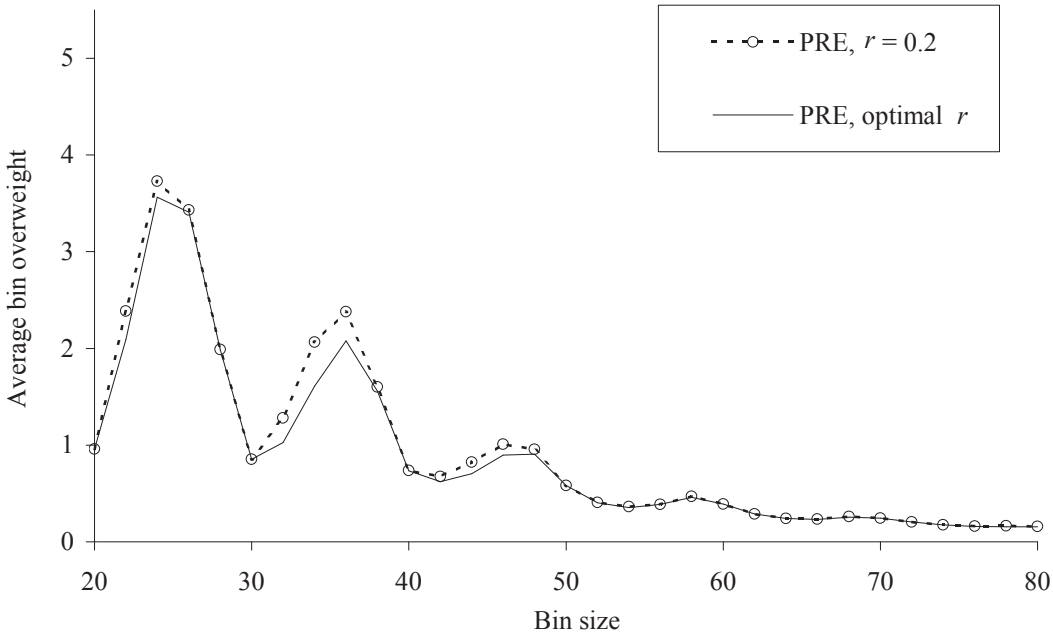


Figure 6.19: Effects of varying bin size on the PRE algorithm for three active bins and the $N_D(10,1.5)$ distribution when using constant $r = 0.20$.

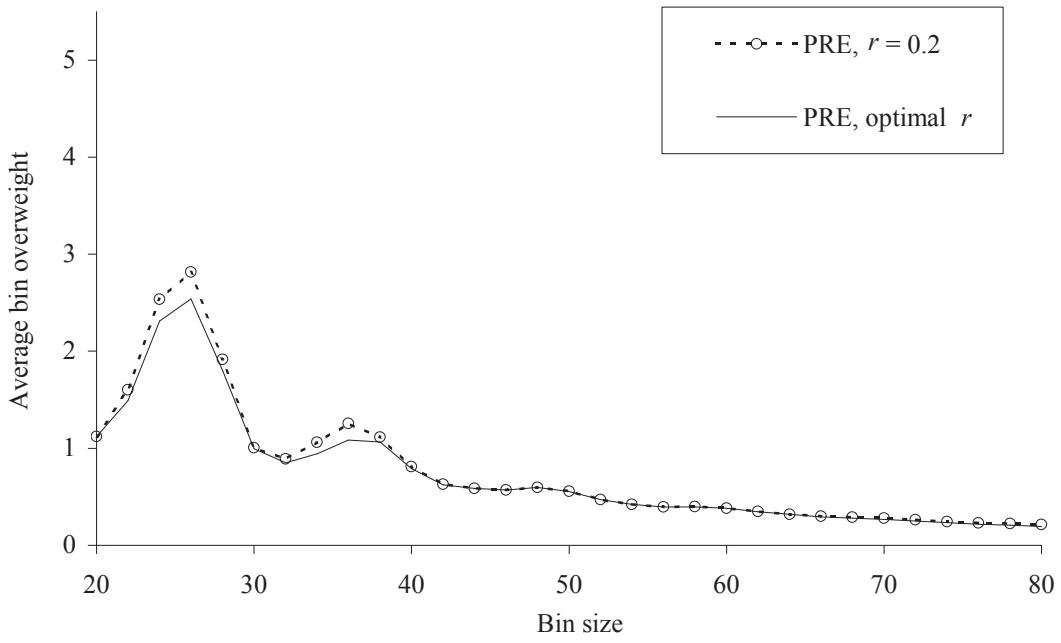


Figure 6.20: Effects of varying bin size on the PRE algorithm for three active bins and the $N_D(10,2.0)$ distribution when using constant $r = 0.20$.

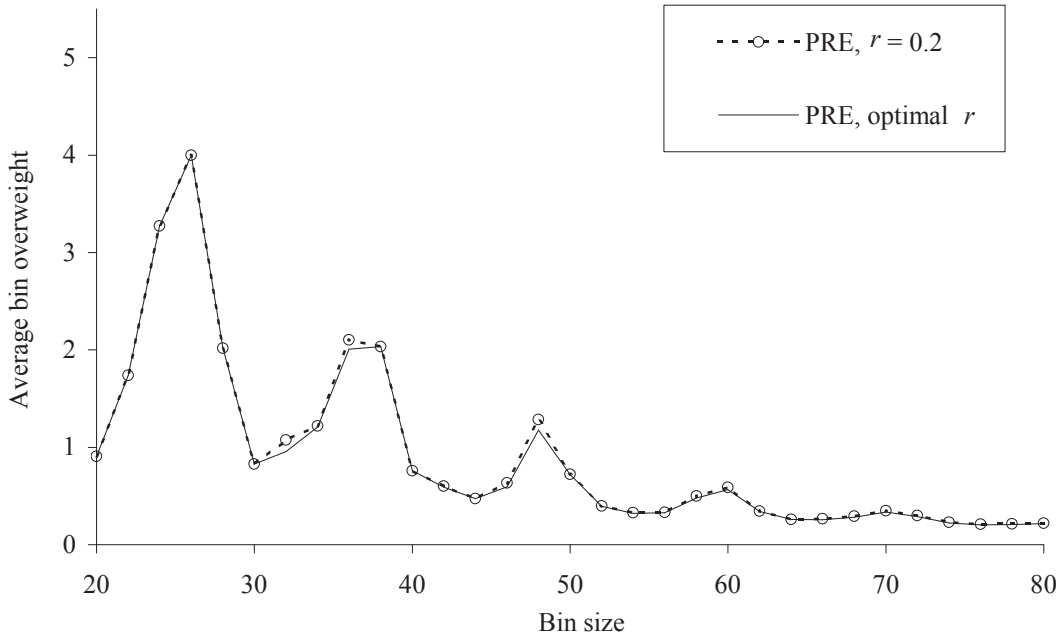


Figure 6.21: Effects of varying bin size on the PRE algorithm for three active bins and the $U_D(8,12)$ distribution when using constant $r = 0.20$.

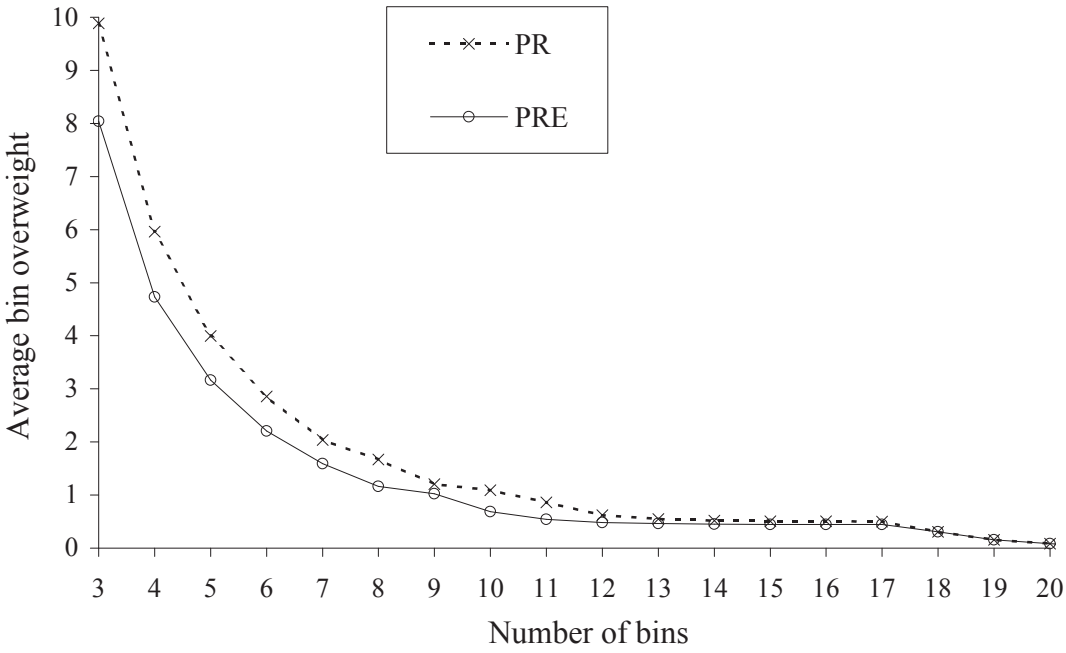


Figure 6.22: Comparison of the performance of the PR and PRE algorithms as a function of the number of active bins, with bin size 600, and the $N_D(100,15)$ distribution.

6.6 MDP Approximations for Bigger Problems

In this section, the Markov Decision Process approach is extended to solve more practically sized problems, i.e., problems with a few hundred different item sizes and up to eight active bins. There are two different bin covering algorithms proposed, namely, MDP2-N and MDP2+. Both have the potential to beat the PR+ algorithm, but only for problems of limited size, and hence they are not suitable as a general replacement of the PR+ algorithm. The PR+ algorithm is used for comparison since it usually does slightly better than the original Prospect Algorithm, and runs faster than the PRE algorithm from the previous section.

The idea is to use the relative cost-to-go function from an MDP with two bins (denoted here as $V_2(s_1, s_2, w)$ for clarity) as an estimator for the cost-to-go function for a problem with k bins, where k is an even number. This is a standard approach [81] when trying to extend MDP methodology to problems with state spaces that are too big to obtain the optimal cost-to-go function. The two-bin problem is chosen for practical reasons, namely, the optimal solution can still be calculated for close to practically sized problems, whereas the optimal solution for the three-bin problem can only be calculated for limited distributions such as the N_D(10,1.5) distribution used in Section 6.3.

In order to make the relative cost-to-go functions more manageable, the state space is reduced by eliminating the next item from the state space by taking an expected value with respect to the item distribution as in the following formula, where $f_{\min}$ and $f_{\max}$ are the minimum and maximum items, respectively:

$$V_2(s_1, s_2) = \sum_{w=f_{\min}}^{f_{\max}} f(w) V_2(s_1, s_2, w) - Q. \tag{6.15}$$

The constant Q depends on the reference state. Assuming that the bin states of the reference state are the same as before, i.e., $\mathbf{s}^* = \mathbf{0}$ (the selected item weight of the

reference state of the extended model does not matter), the value of Q is given by the following equation:

$$Q = \sum_{w=f_{\min}}^{f_{\max}} f(w)V_2(0,0,w). \tag{6.16}$$

The cost-to-go function $V(\mathbf{s})$ is estimated by solving an optimization problem where the k bins are paired, so that the sum of the relative cost-to-go functions from the two-bin MDP problems for all the bin pairs, $V_2(s_1, s_2)$, is minimized. (We use a sum to estimate $V(\mathbf{s})$ for simplicity.) The values for $V_2(s_1, s_2, w)$ for all ordered bin states s_1, s_2 and item sizes w are calculated as in Section 6.3 (the initial cost-to-go vector was set to all zeros and the relative value iteration was run until the maximum error was below 10^{-12}), and the cost-to-go functions for the reduced state space are computed using Equations (6.15) and (6.16), and stored to be used in the MDP approximation.

Formally the optimization problem in state $\mathbf{s}$ is defined as:

$$V(\mathbf{s}) = \quad \text{minimize} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{i,j}(\mathbf{s})V_2(s_i, s_j)$$

subject to

$$\sum_{i=1}^{n-1} x_{i,n}(\mathbf{s}) + \sum_{j=n+1}^{k} x_{n,j}(\mathbf{s}) = 1, \quad \forall n \in 1, 2, \ldots, k,$$

$$x_{i,j}(\mathbf{s}) \in \{0, 1\} \qquad \text{for } 1 \leq i < j \leq k,$$

where

$$x_{i,j}(\mathbf{s}) = \begin{cases} 1 & \text{if bins } i \text{ and } j \text{ are matched,} \\ 0 & \text{otherwise,} \end{cases}$$

$V_2(s_i, s_j)$ is the cost-to-go function from the two-bin MDP,

k is the number of bins,

$\mathbf{s} = (s_1, s_2, \ldots, s_k)$ is the ordered list of bin states of the k-bin problem. (6.17)

This optimization problem has to be solved each time an item is to be allocated to a bin, since the state space $\mathcal{S}$ is so huge that it is not possible to solve it beforehand

for all states in the state space. We solve the optimization problem with complete enumeration, which is easily feasible if the number of bins is limited to eight.

The optimization problem of Equation (6.17) can also be modeled as a (non-bipartite) graph, where the bins are the nodes and the $V_2(s_i, s_j)$ are interpreted as edge weights. The graph is *complete* since any two nodes are connected with an edge, and the objective of the optimization is to find a *minimum weight perfect match*. A perfect match is a set of $k/2$ edges that connect all nodes in pairs, which can of course only be done if k is a even number, since otherwise one node will be left out.

The bin selection for a new item, at each decision epoch t, is done by solving the following optimization problem, where a is the selected bin, $r_t([\mathbf{s}, w], a)$ is the instant reward for placing the next item w in bin a, (see Equation (6.6)), and $\mathbf{s}_{w,a}$ is the resulting bin state:

$$\min_{a \in \{1,2,\ldots,k\}} r_t([\mathbf{s}, w], a) + V(\mathbf{s}_{w,a}). \tag{6.18}$$

Even though we assumed that the number of bins k is even, a problem with an odd number of bins k' can be easily adapted to the optimization problem (6.17). In particular, an empty dummy bin is added to the front of the bin state vector $(s_1, s_2, \ldots, s_{k'})$. This dummy bin cannot receive items and will therefore always stay empty. The modified system will have an even number of bins $k = k' + 1$, with state $\mathbf{s} = (0, s_1, s_2, \ldots, s_{k'})$, and can be solved using the same method as a regular problem with an even number of bins.

There is one fundamental feature of the two-bin cost-to-go function $V_2(s_1, s_2)$ that needs to be mentioned. It receives all items, meaning that the item distribution it "sees" is just the input item distribution $f(w)$. However, in the k-bin setup, each two-bin group is competing for the items, and the items it will receive will not necessarily be distributed as $f(w)$. In particular, each particular combination of bin states is more likely to see items that will fit well into either of its bins and result in a bin with low overfill. In other words, the item distribution it sees is dependent on s_1 and

s_2, and can be expressed as $\hat{f}(w, s_1, s_2)$.

In order to try to correct for this effect, we estimate $\hat{f}(w, s_1, s_2)$, and substitute that for $f(w)$ in Equations (6.15) and (6.16) to yield a modified cost-to-go function:

$$\hat{V}_2(s_1, s_2) = \sum_{w=f_{\min}}^{f_{\max}} \hat{f}(w, s_1, s_2)[V_2(s_1, s_2, w) - V_2(0, 0, w)]. \tag{6.19}$$

Estimation of $\hat{f}(w, s_1, s_2)$ is done by iterating simulation runs. The first simulation run uses $V_2(s_1, s_2)$ (i.e., $f(w)$); the subsequent simulations use an estimate on $\hat{f}(w, s_1, s_2)$ calculated from the following two statistics that are kept for each not-filled single bin state s:

1. A histogram is kept to log the frequency of each item size that is added to a bin. This histogram is denoted $h(w, s)$, where s is the weight in the bin, and w is the size of an added item. Once the iteration run is done, the histogram is normalized and the result is a set of distribution estimates $\hat{f}(w, s)$.

2. A count of the times each state is hit (in any bin) is kept as well. This is denoted $bc(s)$.

After a simulation run, $\hat{f}(w, s_1, s_2)$ is estimated for the next simulation run by plugging the collected data into the following equation:

$$\hat{f}(w, s_1, s_2) = \begin{cases} \dfrac{bc(s_1)\hat{f}(w, s_1) + bc(s_2)\hat{f}(w, s_2)}{bc(s_1) + bc(s_2)} & \text{if } bc(s_1) + bc(s_2) > 0, \\ f(w) & \text{else.} \end{cases} \tag{6.20}$$

Equation (6.20) is simply a weighted sum of the $\hat{f}(w, s)$ estimates for the two bin states, with the weights proportional to how frequently each state was hit. There is a positive probability that neither of the two states in question was hit in a simulation run ($bc(s_1) + bc(s_2) = 0$), and in that case the item distribution is used. This is a rare event that does not materially affect the outcome.

This new algorithm is called MDP2-N, where N denotes the iteration number for the $\hat{f}(w, s_1, s_2)$ distribution estimate, and is listed here:

Algorithm 6.1 (MDP2-N)

Step 1: Set $N = 0$, and $\hat{V}_2(s_1, s_2) = V_2(s_1, s_2)$. Go to Step 2.

Step 2: Run a simulation for the desired number of items. Each item is placed into the bin that minimizes Equation (6.18); the optimization problems from Equation (6.17) are solved on the fly as needed to calculate the $V(\mathbf{s})$ estimates. The statistics $\hat{f}(w, s)$ and $bc(s)$ are kept as described before. If this is the final iteration on N we are done, else go to Step 3.

Step 3: Set $N = N + 1$. A new $\hat{f}(w, s_1, s_2)$ is calculated using Equation (6.20), and using that, a new $\hat{V}_2(s_1, s_2)$ is calculated using Equation (6.19). Go to Step 2.

In the following simulations, the item distribution used is a discrete approximation of a normal distribution with $\mu = 50$ and $\sigma = 9.50$. This distribution is labeled $\mathrm{N}_D(50,10)$ (i.e., we round σ to an integer value for the distribution label). The distribution is described in Appendix A.5. It is not practical to use the same distributions as in Chapters 3 and 4 since then it would take too long to solve the 2-bin MDP problems. Each simulation is run for 31,000 filled bins for each value of N, the first 1000 for warmup, and the rest split into 30 batches in order to calculate the average overfill and a 95% confidence interval for it. The statistics needed for the $\hat{f}(w, s_1, s_2)$ estimate are kept for all the items excluding the warmup.

In Figure 6.23 the average overweight and the confidence interval bounds are plotted as a function of the number of iterations runs for estimating $\hat{f}(w, s_1, s_2)$. Initially, the overfill is about 2.8 and it drops to about 1.7 after the first iteration. Subsequent iterations do not improve the results, and the overfill stays in the 1.6–1.8 range. This is a typical result of doing repeated iterations; there is not much gain

after the first iteration, so in the remainder of simulations only MDP2-0 and MDP2-1 were used.

In Figure 6.24, the average overfills of the MDP2-0, MDP2-1, and PR+ algorithms are compared for bin size 200 and varying numbers of active bins. From this figure we make the following observations:

- MDP2-0 performs best for 3 and 4 bins. This can be explained by the fact that there are only 1 or 2 extra bins compared to the base 2-bin problem, so the cost-to-go estimation in Equation (6.17) is not too bad, even if one uses $\hat{f}(w, s_1, s_2) = f(w)$ (as in Equation (6.15)).

- MDP2-1 equals or beats MDP2-0 when there are 4 bins or more. Here the improved estimation of $\hat{f}(w, s_1, s_2)$ is starting to pay off.

- PR+ is similar to MDP2-1 for 5 and 6 bins and is the most competitive algorithm for 7+ bins. The reason might be that as the number of bins increases, the accuracy of the $V(\mathbf{s})$ estimation decreases, making MDP2-N less competitive.

In the MDP2-N algorithm, the focus is on estimating $\hat{f}(w, s_1, s_2)$ in order to improve the cost-to-go function estimator in Equation (6.17). We expect that the result of this is to reduce the value of the cost-to-go function, since each 2-bin combination is more likely to receive a new item of a given weight if the resulting cost-to-go value is low rather than high. The iterations needed in MDP2-N are cumbersome and do not seem to add much. Therefore it would be advantageous to eliminate them.

An alternate way is to focus on the effect of the iterations on the cost-to-go function estimator directly, thereby making the iterations unnecessary. This can be achieved by adding a discount multiplier, $\alpha \in (0, 1]$, on the second term in Equation (6.18):

$$\min_{a \in \{1,2,\ldots,k\}} r_t([\mathbf{s}, w], a) + \alpha V(\mathbf{s}_{w,a}). \tag{6.21}$$

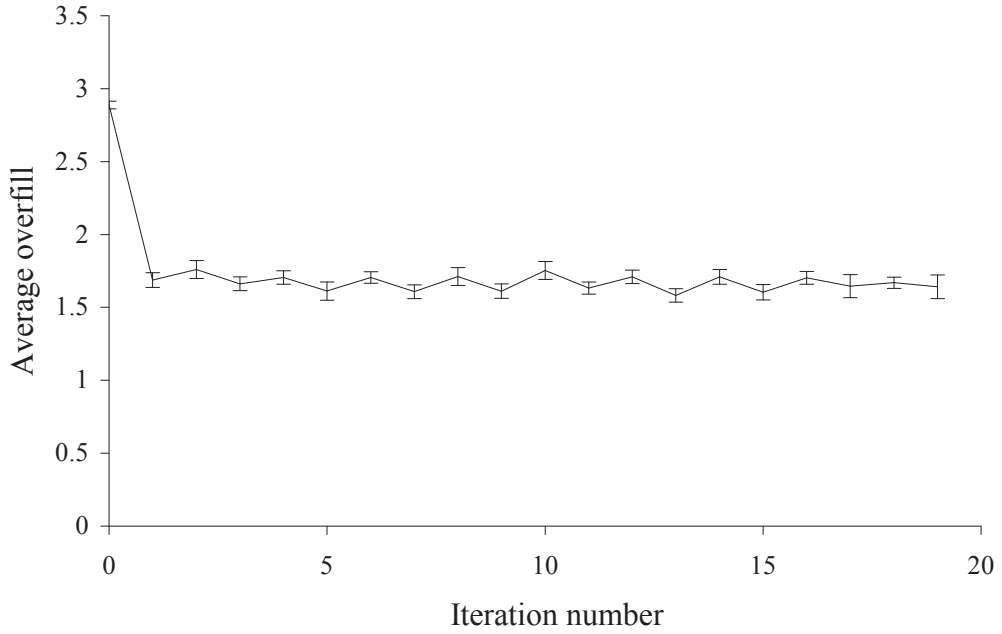


Figure 6.23: Effect of iterating the $\hat{f}(w, s_1, s_2)$ estimate for eight active bins, bin size 200, and the $N_D(50,10)$ distribution. 95% confidence intervals are shown at each point.

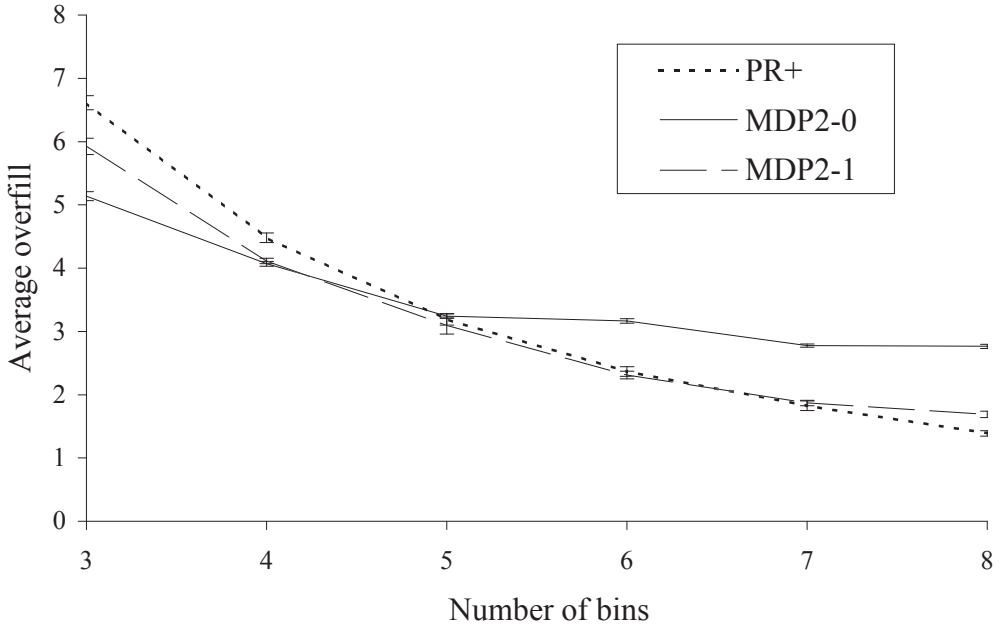


Figure 6.24: Effect of the number of active bins on the MDP2-0, MDP2-1 and PR+ algorithms for bin size 200 and the $N_D(50,10)$ distribution.

The idea is to directly discount the cost-to-go estimate, since it is only a rough estimate of the real cost-to-go function, and hence is not as accurate as the reward term of the equation. However, straight multiplication by the discount multiplier (as in Equation (6.21)) turns out to be dangerous, because it can cause a bin to freeze up almost full, usually with just a space for an item of size one to fill it. The bin is frozen because it can only be filled with high overfill (i.e., a high value for $r_t(\mathbf{s}, a)$ in Equation (6.21)), and hence it is unlikely that it will receive the next item. A solution to this problem is to use the discount multiplier more selectively by employing it directly in Equation (6.17), and only let it apply when the bins have more than $f_{\min}$ left in them. First define the function $\beta(\alpha, s_1, s_2)$ as:

$$\beta(\alpha, s_1, s_2) = \begin{cases} \alpha & \text{if } b - s_1 \geq f_{\min} \text{ and } b - s_2 \geq f_{\min}, \\ 1 & \text{else} \end{cases} \tag{6.22}$$

(recall that b is the bin size). Then use it in the cost-to-go estimator as follows

$$\begin{aligned} V(\mathbf{s}, \alpha) = \text{minimize} & \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{i,j}(\mathbf{s}) \beta(\alpha, s_i, s_j) V_2(s_i, s_j) \\ \text{subject to} & \sum_{i=1}^{n-1} x_{i,n}(\mathbf{s}) + \sum_{j=n+1}^{k} x_{n,j}(\mathbf{s}) = 1, \quad \forall n \in 1, 2, \ldots, k, \\ & x_{i,j}(\mathbf{s}) \in \{0, 1\} \qquad\qquad\qquad \text{for } 1 \leq i < j \leq k. \end{aligned} \tag{6.23}$$

The effect of the β function is to give pairs with at least one nearly full bin higher cost-to-go estimate, and thereby making them less likely in the selection, since the final cost-to-go value is estimated with minimization. The selection of α is critical, but note that there is information in the algorithm that can be utilized to make the selection easier. It is known what the average expected overfill for the 2-bin MDP, $\bar{o}_2$, is, and as the new algorithm is run, its average overfill, $\tilde{o}$, can be measured, for example by calculating a running exponential moving average. By trial and error we found out that defining α as twice the ratio between those two numbers ($\tilde{o}$ and $\bar{o}_2$) works well, since it is a measure of how much better (in terms of average overfill) the

approximate algorithm is doing compared to the base 2–bin MDP (in this case the value for α is time-varying). The value of the discount rate α also has to be capped at 1. The formula for the selection of α is then:

$$\alpha = \min\left(\frac{2\tilde{o}}{\bar{o}_2}, 1\right). \tag{6.24}$$

This new algorithm is called MDP2+. To measure $\tilde{o}$, exponential smoothing of the running average of the overfill of finished batches is used, with 0.01 as the smoothing constant. Figure 6.25 compares MDP2+ to the algorithms of Figure 6.24. It turns out that the MDP2+ algorithm ties MDP2-0 for three bins but beats it for four or more bins, yields better results than MDP2-1, and beats the PR+ algorithm in all cases. In Figures 6.26 to 6.31, MDP2+ is compared to the PR+ algorithm for all bin sizes from 100 to 250 (in increments of 10), with one figure for each different number of bins, $k \in \{3, 4, \ldots, 8\}$. Apart from the varying bin size, the simulations are done in the same fashion as the ones for Figure 6.24. In all cases, MDP2+ has significantly lower overfill than the PR+ algorithm. In Table 6.10 the average overfill for the algorithms as a function of the number of bins is listed, showing that MDP2+ has on average 81.9% of the PR+ algorithm's overfill (4.97 vs. 6.07). The table also shows how the relative difference in the overfill reduces with the number of bins. The reason for that might be that as the number of bins increases, the accuracy of the $V(\mathbf{s})$ estimation decreases, making MDP2+ less competitive.

The conclusion of this section is that it is possible to expand the optimal MDP strategy to handle problems of more-realistic size than in Section 6.3, and the proposed algorithms put forth can outperform the PR+ algorithm in at least some instances. In our simulations, MDP2+ is better than both MDP2-0 and MDP2-1. However, these approximate MDP approaches do not scale up to practical-size problems because of their exponential rise in memory requirements and computation time as the size of the problem grows.

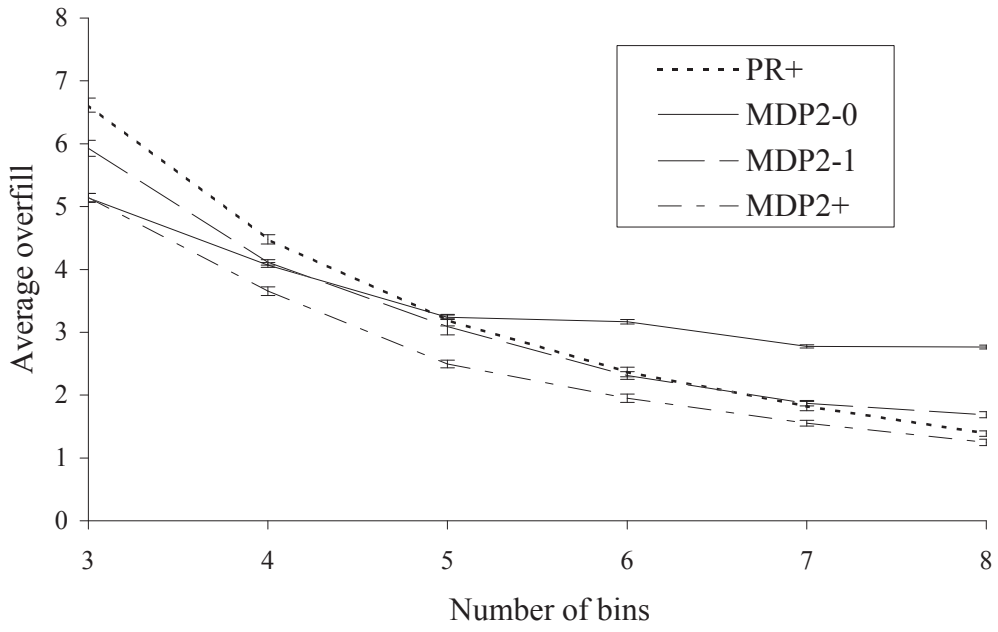


Figure 6.25: Effect of the number of active bins on the MDP2, MDP2+, and PR+ algorithms overfill for bin size 200 and the $N_D(50,10)$ distribution.

Bins	MDP2+	PR+	Ratio
3	7.09	9.09	78.0%
4	5.77	7.12	81.0%
5	4.81	5.97	80.6%
6	4.36	5.22	83.5%
7	4.02	4.69	85.7%
8	3.78	4.31	87.8%
Average	4.97	6.07	81.9%

Table 6.10: Summary of the average overfills of MDP2+ and PR.

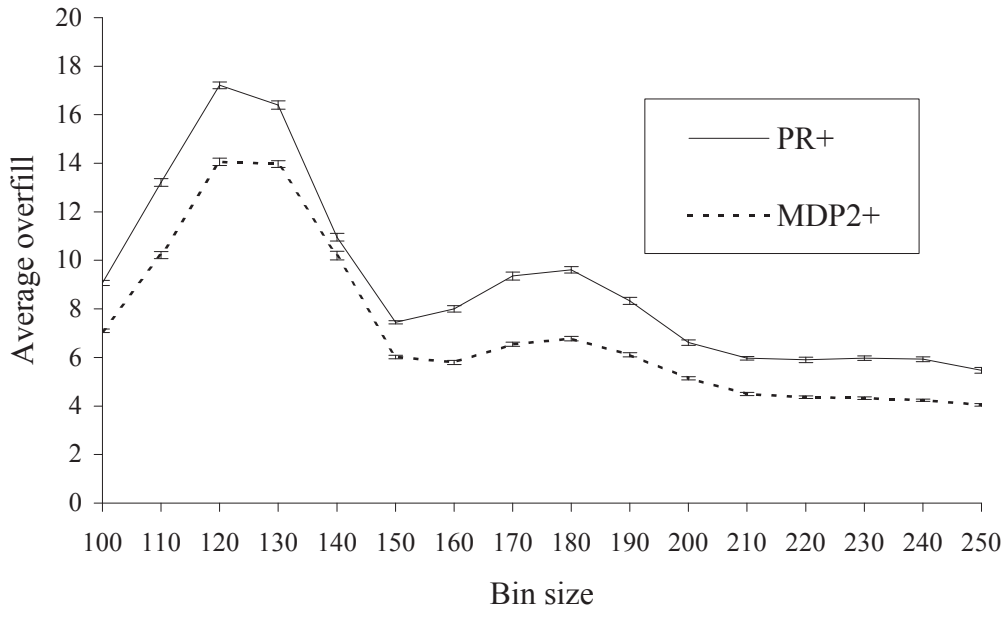


Figure 6.26: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, three active bins, and the $N_D(50,10)$ distribution.

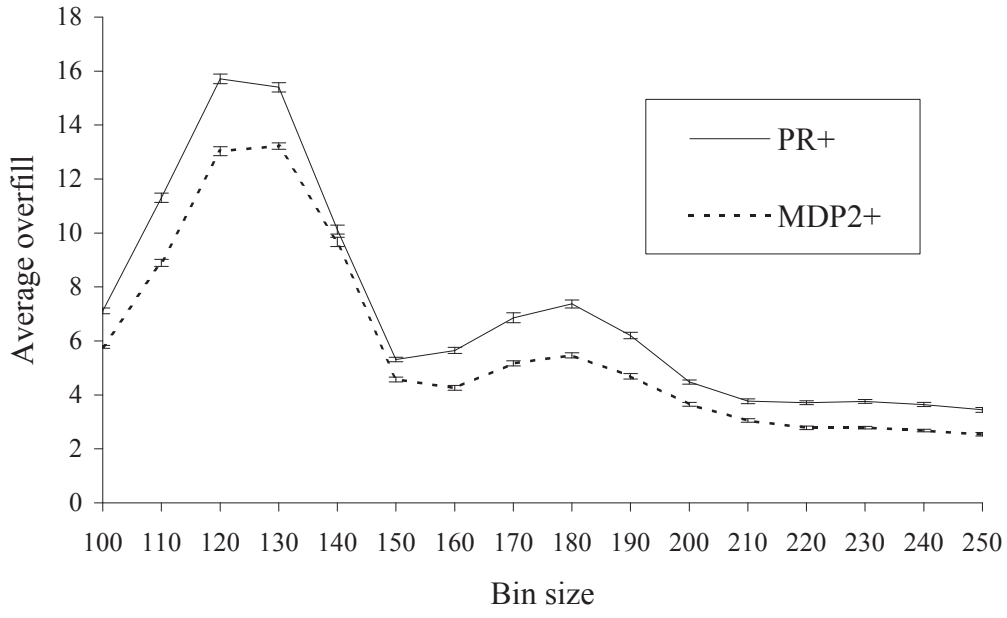


Figure 6.27: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, four active bins, and the $N_D(50,10)$ distribution.

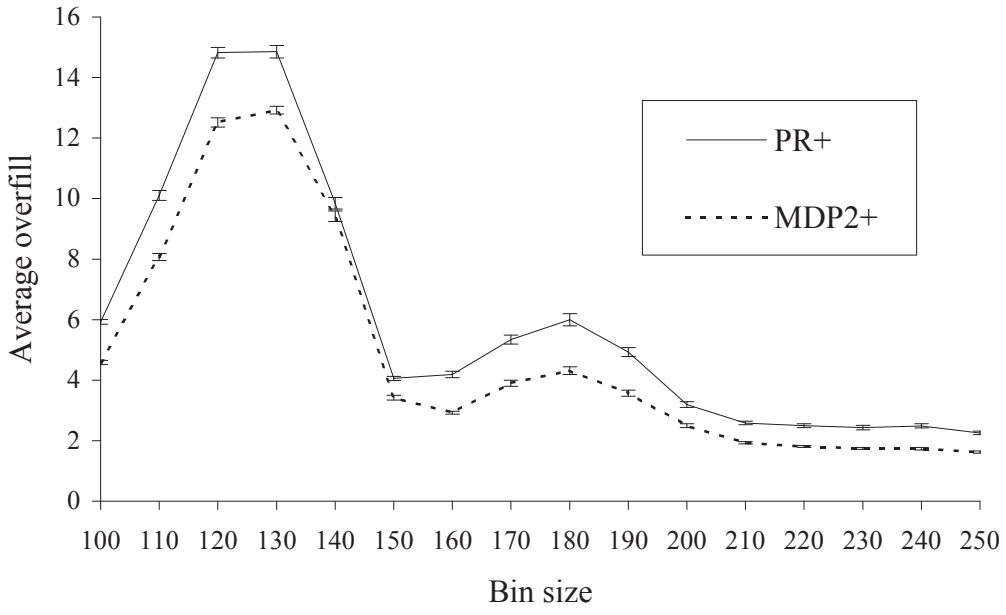


Figure 6.28: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, five active bins, and the $N_D(50,10)$ distribution.

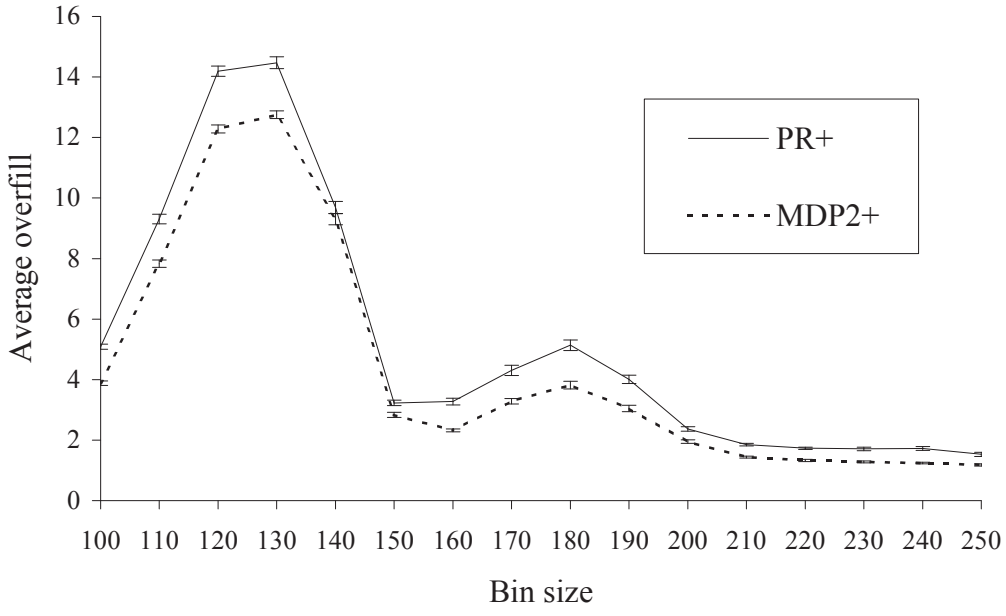


Figure 6.29: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, six active bins, and the $N_D(50,10)$ distribution.

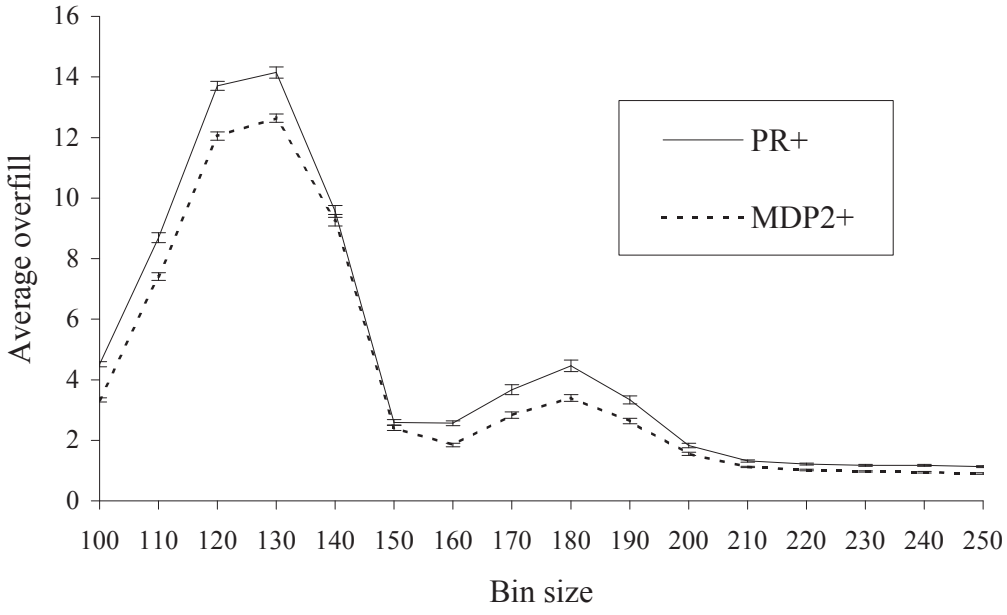


Figure 6.30: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, seven active bins, and the $N_D(50,10)$ distribution.

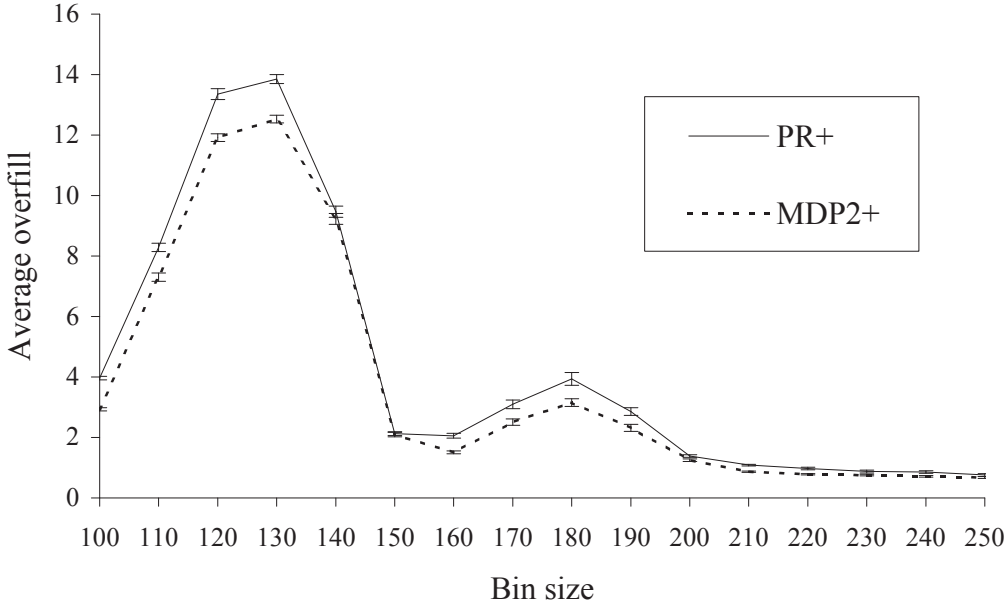


Figure 6.31: Comparison of the MDP2+ and PR+ algorithms' overfill for varying bin sizes, eight active bins, and the $N_D(50,10)$ distribution.

6.7 Rollout Prospect Algorithm

In this section, a rollout algorithm for bin covering is tested. Rollout is an MDP-inspired add-on to some heuristic solution algorithms that can yield an improvement in performance. For background and definitions of rollout algorithms, Bertsekas et al. [15, 16] and Galerpin and Tesauro [114] are recommended. The main strength of the rollout algorithm is that is applies to practically sized problems.

Rollout works as follows:

Definition 6.5 (Rollout) For a heuristic algorithm $u = \pi(x)$, where u is the action in state x, an improved *rollout* algorithm $u = \pi_r(x)$ is defined as:

- For each possible action u in state x, the expected cost of taking u in x followed by the heuristic algorithm π is evaluated.
- $\pi_r(x)$ is the action that minimizes the expected cost.

In the case of bin covering, the evaluation is done by running repeated short Monte Carlo simulations. The simulations are short since the effects of selecting a bin for a single item are transient — the cost regresses to the long-term cost of the Heuristic Algorithm after a few bins have been filled. Formally, the *Rollout Prospect Algorithm* is defined as:

Algorithm 6.2 (Rollout Prospect Algorithm)

Step 1: Pick the next item and test-fit it into all active bins. After each test-fit, run a series of simulations using the PR+ Algorithm to estimate the future overfill resulting from the test-fit. The item data for the simulation is the same as that which the Prospect Function is calculated from, and the number and length of simulation runs for each test fit are parameters that need to be set. Go to Step 2.

Step 2: Put the item into the bin with the lowest estimated overfill. If the selected bin gets filled, then close it and activate a new empty bin. Go to Step 1.

Relative to the PR+ Algorithm, the rollout addition better takes into account the presence of multiple bins. The rollout algorithm is very simple to program, and adds little complexity to the code. Running these Monte Carlo simulations is bound to be computationally expensive, so it is most practical to use it for heuristic algorithms that are fast. It should be noted though that the Monte Carlo simulations are very easy to speed up on multi-processor or multi-core computers, since it is simple to parallelize the computations. There are two parameters that need to be set, namely, the number of bins that are filled in each simulation and the number of simulation repeats. The product of these two parameters determines how much computational effort goes into the rollout calculations.

To demonstrate the Rollout Prospect Algorithm's potential (Rollout PR+ for short), a few simulations are run. The rollout is added onto the PR+ Algorithm from Section 5.5, and it is tested using the $N_D(100,15)$ distribution (see Appendix A.3). There are 2000 "real" bins filled in each simulation (for each point plotted in Figures 6.32–6.37), and for each item that is added to a real bin, the rollout simulations are performed as well, filling many more "virtual" bins. First the rollout simulation length parameters, namely the number of simulation repetitions and filled bins per simulation, are examined. In those simulations, the Prospect zone parameter is kept constant at 10. The first series of tests is done with four active bins. Figure 6.32 shows how the number of rollout simulations repeats affects the overweight when the number of filled "virtual" bins per rollout simulation is held constant at four, and the rollout repeats vary from 200 to 1400, for a total of 800 to 5600 filled virtual bins simulated per each real item. As the number of rollout repeats increases from 200, the overfill drops until the repeats reach 1000. Above 1000 the overfill starts to oscillate, indicating that there is little to gain in additional rollout simulation

repeats. This is investigated further in Figure 6.33, where the number of repeats is increased to 100,000 per real item. The average overfill does not drop further, even if the number of repeats is increased greatly, indicating that there is little to gain from increasing repeats above some point. Note that using so many repetitions is not practical, because it takes too long to calculate a decision on each item (about 10 seconds on a computer with an Intel Core i7 2640 2.8 GHz processor in the case of 100,000 repeats). A likely reason for the oscillating behavior for large numbers of rollout repeats is that the effect of increasing rollout repeats is so weak that many more real items per simulation are needed to show it. However, the simulations are time consuming, and hence this issue is not probed further in this thesis.

In Figure 6.34, the number of bins filled in each rollout simulation is analyzed, with the total number of virtual bins filled in all rollout simulation repeats being kept constant at 4000, and the number of filled bins per rollout simulation repeat varied from 2 through 10 in steps of 2. Therefore, the number of rollout simulation repeats starts at $4000/2 = 2000$ and ends at $4000/10 = 400$. The optimal value for the number of filled bins in each rollout simulation repeat is 4. Finally, for the four active bins case, Figure 6.35 shows how the PR+ Algorithm zone parameter affects overfill. For PR+, the optimal value of the zone is 11, and for the Rollout PR+, the optimal value of the zone is 10. Rollout PR+ has in all cases lower overfill, and is less sensitive to the zone parameter as well.

The second set of simulations is done for eight active bins. In these simulations, the total number of virtual bins filled in each rollout simulation series is 38400. This number was determined with pre-testing; it is large enough for the Rollout PR+ Algorithm to beat the PR+ Algorithm, but small enough for the simulation of filling 2000 real bins to be reasonably fast, and capable of finishing in a few hours. Note that the number of rollout simulations is almost ten times more than in the four-bin case before. Thus the number of rollout simulations needed to improve on the base

algorithm appears to increase non-linearly as the number of bins grows. In Figure 6.36 the number of bins filled in each rollout simulation is analyzed, varied from 8 through 32 in steps of 8 first and then a jump of 32 for the last simulation. The optimal value for the number of filled bins in each rollout simulation repeat is 24 (six times larger than for four bins). Figure 6.37 shows how the Prospect Algorithm zone parameter affects overfill. For both algorithms the optimal zone value is 5, and again Rollout PR+ has in all cases lower overfill, and is less sensitive to the zone parameter as well.

The conclusion is that a rollout algorithm add-on to the Prospect Algorithm is feasible, simple to code, and can improve performance. How practical the rollout is depends, however, on the computing power available to run the bin covering algorithm; i.e., is there enough processing power available to run the necessary rollout simulations? The parameters of the rollout simulations need to be tested thoroughly before deployment since, as the above tests show, good choices of the total number of virtual bins needed to be filled in the rollout simulations vary with problem parameters (this is true of both the number of virtual bins per repeat and the number of repetitions), and appear to increase greatly with the number of active bins available.

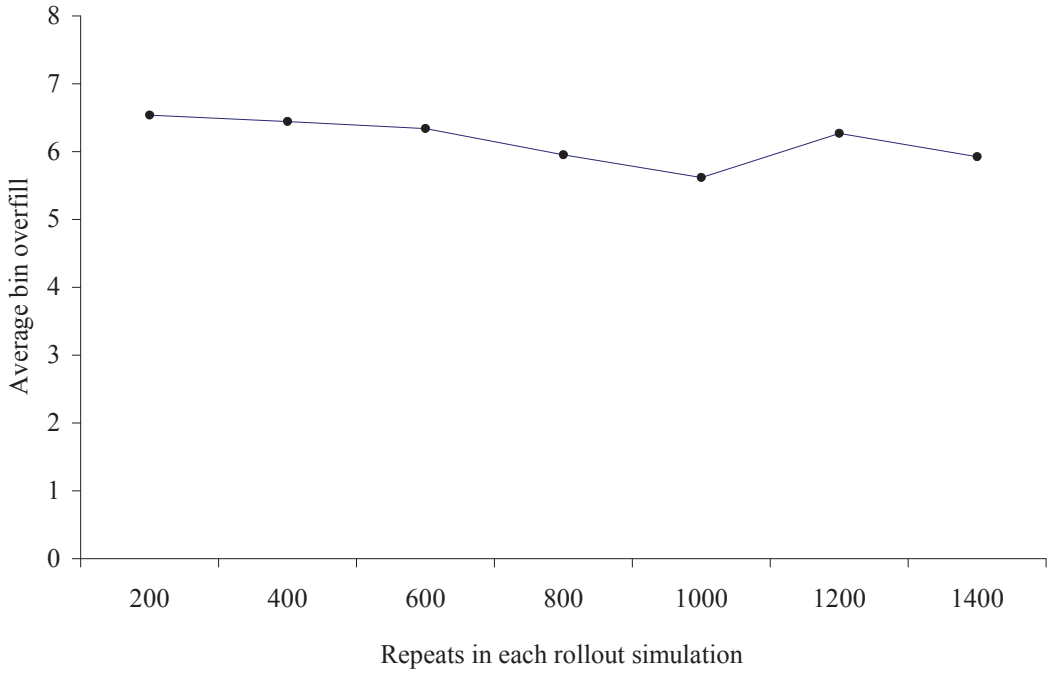


Figure 6.32: Rollout Prospect Algorithm parameter test I. Data is $N_D(100,15)$, four bins, zone parameter 10, bin size 400, and four virtual bins per rollout simulation.

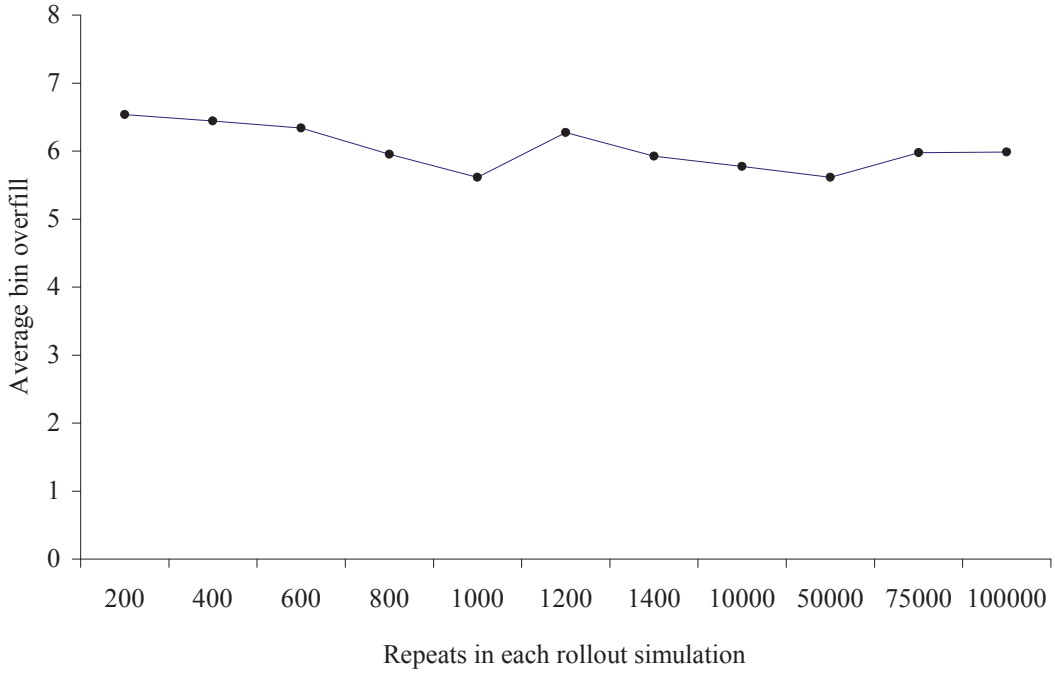


Figure 6.33: Rollout Prospect Algorithm parameter test II. Data is $N_D(100,15)$, four bins, zone parameter 10, bin size 400, and four virtual bins per rollout simulation.

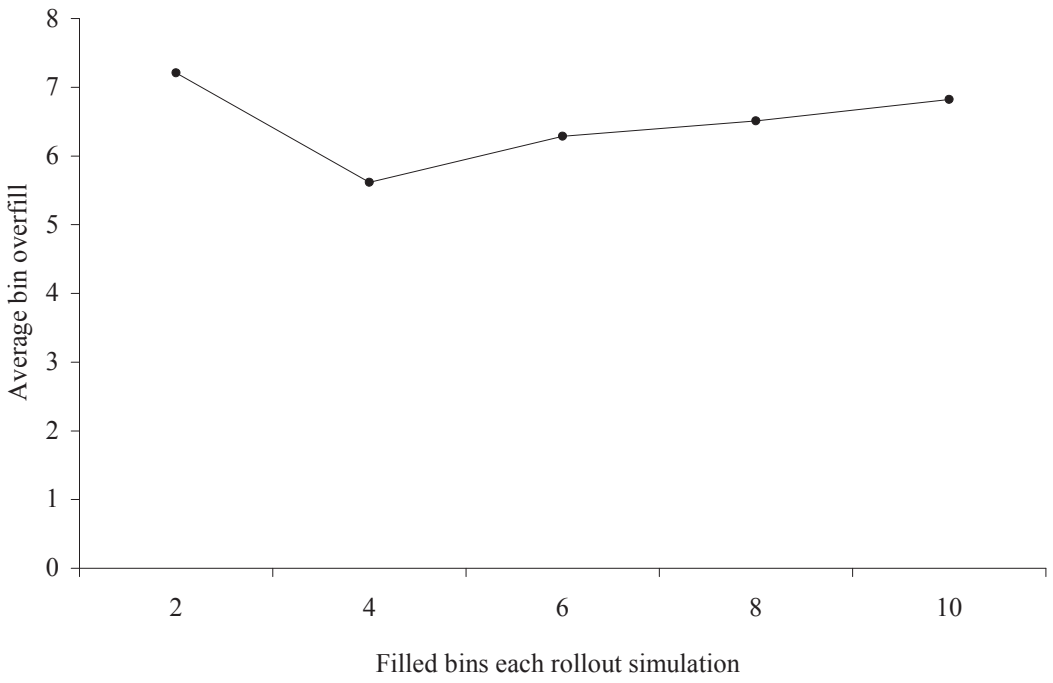


Figure 6.34: Rollout Prospect Algorithm parameter test III. Data is $N_D(100,15)$, four bins, zone parameter 10, bin size 400, and 4000 virtual bins filled over all repetitions.

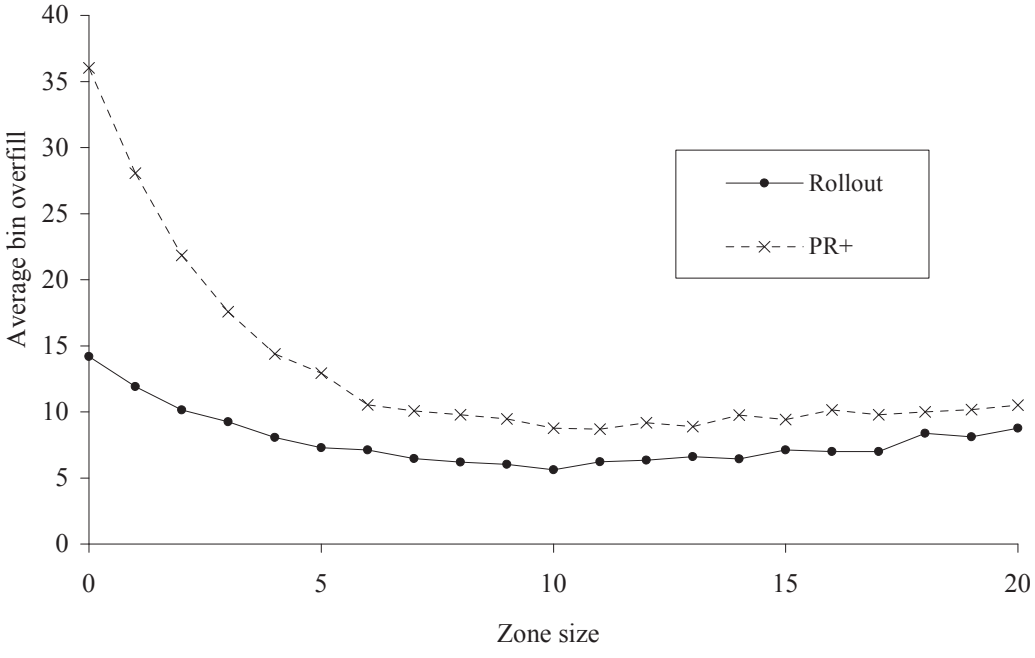


Figure 6.35: Zone parameter sensitivity of the PR+ Algorithm and Rollout PR+ Algorithm. Data is $N_D(100,15)$, four bins, bin size 400, Rollout Prospect Algorithm repetitions 1000, and bins per simulation 4.

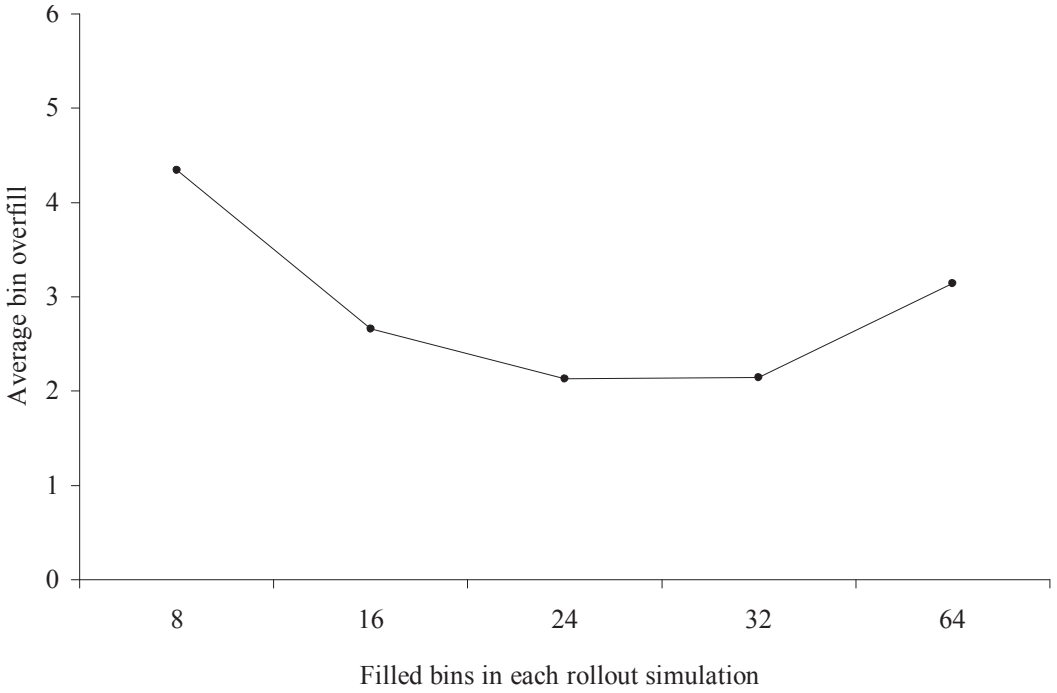


Figure 6.36: Rollout Prospect Algorithm parameter test IV. Data is $N_D(100,15)$, eight bins, zone parameter 10, bin size 400, and 38400 virtual bins filled over all repetitions.

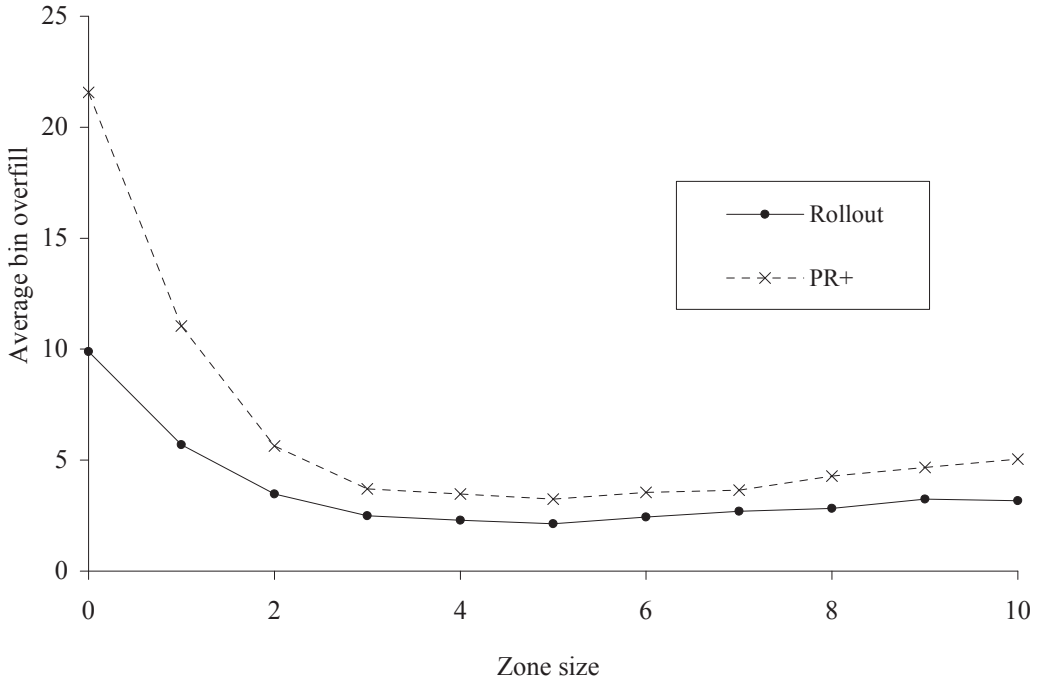


Figure 6.37: Zone parameter sensitivity of PR+ Algorithm and Rollout PR+ Algorithm. Data is $N_D(100,15)$, eight bins, bin size 400, Rollout Prospect Algorithm repetitions 1600, and bins per simulation 24.

CHAPTER 7

ON-LINE BIN-COVERING WITH LOOKAHEAD

7.1 Introduction

In this section, two types of relaxations of the on-line constraint are examined. The motivation for these relaxations is that they correspond to a different type of machine than the one that is described in Chapter 1. The relaxations allow the algorithm to *look ahead* into the oncoming stream of items, so that the size of more than just the next item is known. According to Coffman et al. [25], this relaxation can be classified as *Semi-on-line*. To solve these relaxed problems, two different algorithms based on solving the *Subset-Sum* problem and a modified version of the Prospect Algorithm are applied. The performance of these algorithms is compared to the Prospect+ Algorithm.

The outline of this chapter is as follows. In Section 7.2, the Subset-Sum problem is defined, and its solution algorithm described. In Section 7.3, an algorithm for combining items from several static weighing buffers into batches is described and tested. Finally, in Section 7.4, on-line algorithms that know the weight of the next l items are described and tested.

7.2 Binary Knapsack and Minimization Binary Knapsack Problems

The Subset-Sum problem is a special case of the *Binary Knapsack* optimization problem (BinKP). Both problems are NP-hard [94]. The objective of the Binary Knapsack problem is to fill a knapsack of size $c \in \mathbb{N}$ with items having size $a_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$ $(i = 1, \ldots, n)$, so that the total size of the selected items does not exceed the capacity while the profit is maximized [71]. This can be formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i x_i \leq c, \\ & x_i \in \{0,1\}, \quad \text{for all } i = 1,2,\ldots,n. \end{aligned} \tag{7.1}$$

The binary decision variables $x_1, \ldots, x_n$ indicate the selected items. Note that by solving Problem (7.1), the corresponding *Minimization Binary Knapsack* problem (MinBinKP) has also been solved:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} p_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_i y_i \geq d, \\ & y_i \in \{0,1\}, \quad \text{for all } i = 1,2,\ldots,n, \end{aligned} \tag{7.2}$$

where $d = \sum_{i=1}^{n} a_i - c$. The decision variables $y_1, \ldots, y_n$ indicate which items are used in the MinBinKP, which means they are not used in the BinKP. The problems are therefore linked; so for example if $x_1, \ldots, x_n$ is a solution to BinKP, then the corresponding solution to MinBinKP is $y_i = 1 - x_i$, for $i = 1, \ldots, n$, and vice versa. If the profit equals the item size, or $p_i = a_i$ for all $i = 1, \ldots, n$, then Problems (7.1) and (7.2) are denoted the Subset-Sum and Minimization Subset-Sum problems, respectively. The Subset-Sum problem corresponds to a bin-packing problem with bin size c and known item sizes, whereas the Minimization Subset-Sum problem agrees with a bin-covering problem with bin size d and known item sizes.

The Subset-Sum problem can be solved exactly with a dynamic programming algorithm that has time and space complexity $O(nc)$ [94]. If the Minimization Subset-Sum problem is solved instead, then a dynamic programming algorithm that has time and space complexity $O\big(n(d + a_{\max} - 1)\big)$ is needed, where $a_{\max}$ is the largest of the n items. Hence, the values of c, $d = \sum_{i=1}^{n} a_i - c$, and $a_{\max}$ determine which of the two algorithms has smaller time and space complexity. One can therefore save calculation

time by having both dynamic programming algorithms on hand, and selecting the one with the lower complexity. The time and space complexity of such a combined algorithm is:

$$O\big(\min(d + a_{\max} - 1, c)n\big) \tag{7.3}$$

Such a combined dynamic programming algorithm is used as the basic Subset-Sum solver in this chapter.

7.3 The Subset-Sum Problem

The first problem type corresponds to a *static* grader where the items are weighed on static scales, and then kept in buffers until they are selected to go into a bin. The formal definition of the problem is:

Definition 7.1 (*p*-pseudo on-line constraint) The items arrive on-line, but now there are p buffers that can hold one item each. The bins are filled by choosing any subset of those p items, and after each bin has been filled, the empty buffers are refilled on-line with items.

Figure 7.1 shows the scheme for a hypothetical packing machine. The incoming items are pushed from the incoming conveyor into empty weighing buffers. The machine waits for the weighing buffers to fill, and then selects the best combination of items to fill the collecting bin. The selected items are dropped into the collecting bin via the collecting funnel. The filled collecting bins are conveyed away.

An approximation algorithm for this problem is to solve a series of Subset-Sum problems; i.e., to fill one bin at a time from the items in the buffers. However, this is not sufficient, because it does not cover the situation when there are not enough items in the buffers to fill the collecting bin. A simple modification is to use the Next-Fit Algorithm initially as a secondary selection criteria, and then switch to solving a Subset-Sum problem after the empty space can be filled with the available

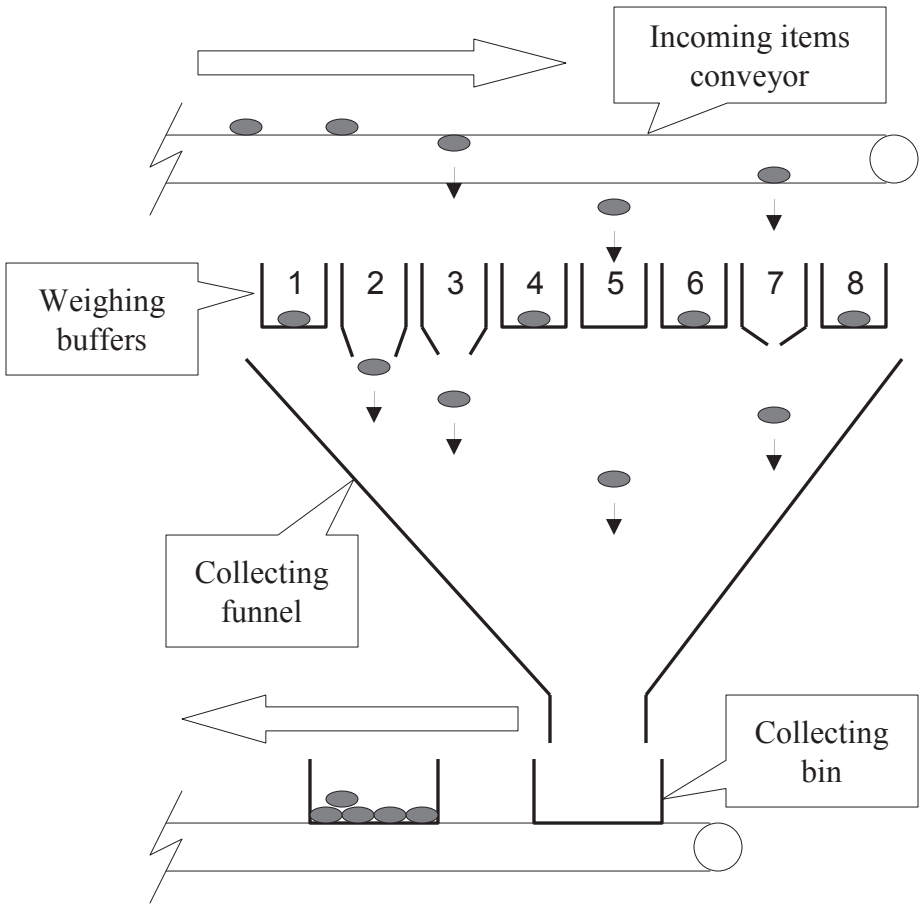


Figure 7.1: Hypothetical Packing Machine.

items. This modification also helps when the size of the available items is not much greater than the empty space in the bin, since in those situations, the number of item combinations that can fill the bin is limited, which leads to increased overfill. To implement this Next-Fit modification, a *fill factor* $\alpha \in (0, 1]$ is introduced. If the total item size in the buffers times α is less than the empty space in the collecting bin, then the oldest item (i.e., the item that has been sitting in a weighing buffer the longest) is added to the bin. If the total item size in the buffers times α is more than the empty space, the bin is filled with items from the buffers. This combined algorithm is called *Subset-Sum NF* (SSNF), and it is defined as follows:

Algorithm 7.1 (Subset-Sum NF Algorithm for Bin-Covering)

Step 1: Wait for all buffers to be filled with items. Let W be the total size of the available items, B be the empty space in the collecting bin, and α be the fill factor.

Step 2: If $\alpha W < B$, then put the oldest item into the collecting bin and go to Step 1. Else go to Step 3.

Step 3: Calculate the optimal combination of items that fill the collecting bin with the least amount of overfill, and put those items into the collecting bin. Empty the collecting bin and go to Step 1.

Algorithm 7.1 can be improved by replacing the initial Next-Fit Algorithm by a modified Prospect Algorithm. Instead of selecting the oldest item in the pre-filling process, select the item that improves the Prospect function of the unfilled collecting bin the most. This combined algorithm is called *Subset-Sum Prospect* (SSP), and it is defined as follows:

Algorithm 7.2 (Subset-Sum Prospect Algorithm for Bin-Covering)

Step 1: Wait for all buffers to be filled with items. Let W be the total size of the available items, B be the empty space in the collecting bin, and α be the fill factor.

Step 2: If $\alpha W < B$, then put the item into the collecting bin that improves the Prospect function of the bin the most and go to Step 1. Else go to Step 3.

Step 3: Calculate the optimal combination of items that fill the collecting bin with the least amount of overfill, and put those items into the collecting bin. Empty the collecting bin and go to Step 1.

To calculate the optimal combination of items in Step 3 of SSNF and SSP, we use a dynamic programming algorithm. Figures 7.2–7.5 show results of simulations of SSNF and SSP. In the first two simulations, there are eight weighing buffers, the item distribution is $\mathrm{N}_D(100,15)$, and the bin size is varied from 200 to 800. Figure 7.2 shows what effects the fill factor α has on performance (α is varied from 10% to 90% in 10% increments, and each simulation is run for 10000 filled bins for each of the 31 bin sizes, for a total of $310,000$ simulations for each data point). The average overfill of each simulation is calculated, and the average over all bin sizes is plotted for both SSNF and SSP@Both graphs are U-shaped, with the curve sloping up when α approaches either 0% or 100%. The algorithms are more sensitive to having α too low, since in that case most of the collecting bin is filled with the simple, pre-fill heuristic, defeating the purpose of choosing an optimal selection of the available pieces to fill the bin. It is therefore not desirable to have α less than 30%, but such cases are included in Figure 7.2 for completeness. For both SSNF and SSP, the optimal value of α is 60%, and that is the value used in the subsequent simulations.

Figure 7.3 compares the performance of SSNF and SSP, both with $\alpha = 60\%$. Each simulation is run for 601,000 finished bins. The first 1000 bins are ignored for warmup,

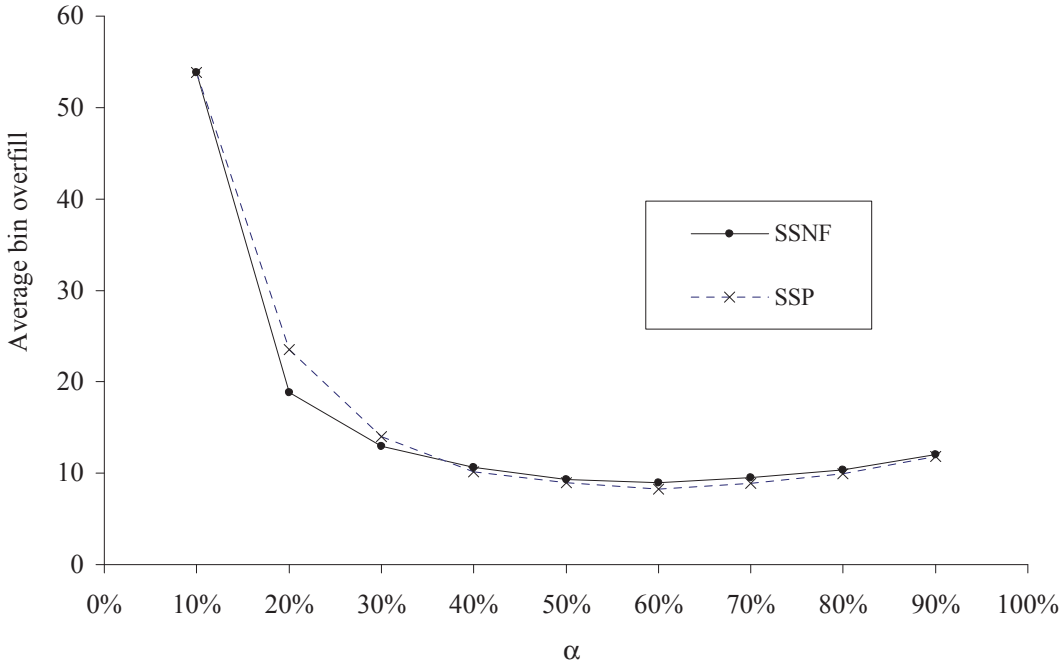


Figure 7.2: Effects of the fill factor α on SSNF and SSP for 8 weighing buffers and the $N_D(100,15)$ distribution.

and then the average overfill for each batch of 20000 finished bins is calculated, for 30 averages in total. We use this data to calculate the average expected overfill and its 95% confidence interval. The 95% confidence intervals are so tight that their widths are about the same as the plot-line, so therefore they are not plotted but just listed in Appendix E. Running enough simulations to obtain a tight confidence interval is very time consuming, and it did not add much insight, so it was not done again in the remainder of this chapter. The performance is identical for bin sizes up to approximately 440 — as expected since the secondary decision criteria does not apply there (because the collecting bin can always be filled from the items in the weighing buffers). For bin sizes above 440, SSP performs better.

The performance of the SSP algorithm depends on the number of weighing buffers, and the Prospect Algorithm depends on the number of collecting bins. Figure 7.4 compares the SSP algorithm with eight and twelve weighing buffers (SSP 8 and SSP 12) to the Prospect+ Algorithm with eight bins. The simulation data and setup are

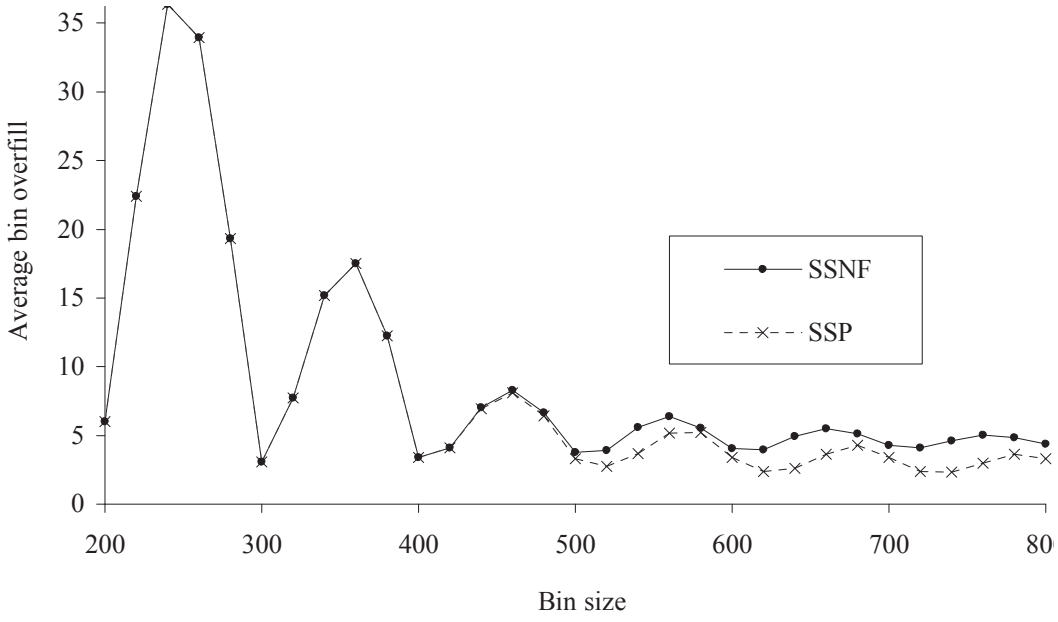


Figure 7.3: Comparison of SSNF and SSP for 8 weighing buffers and the $N_D(100,15)$ distribution.

identical to the previous simulations. For bin sizes below 400, both SSP cases perform better than the Prospect Algorithm. Above 400, the Prospect Algorithm performs better than SSP 8, but not as well as SSP 12. This shows that for this distribution, 8 SSP buffers are not enough to consistently beat PR+ with 8 bins, but 12 SSP buffers are. Figure 7.5 compares SSP and PR+, again for the same simulation data and setup, but now with a fixed bin size of 600 and varying number of bins (PR+) and weighing buffers (SSP). It shows that both algorithms achieve practically zero overfill if they have enough buffers/bins. This comparison cannot be used to determine which algorithm is the "best" since they conform to fundamentally different constraints, constraints that are set by the mechanics of the machine they are programmed for. Nevertheless, the above analysis shows that a good working algorithm exists for both types of machines. There is, however, a difference in the algorithms that does not show in the graphs, namely, that PR+ has a critical parameter, the zone parameter, that needs to be set for each case. SSP only has the pre-fill parameter α that can

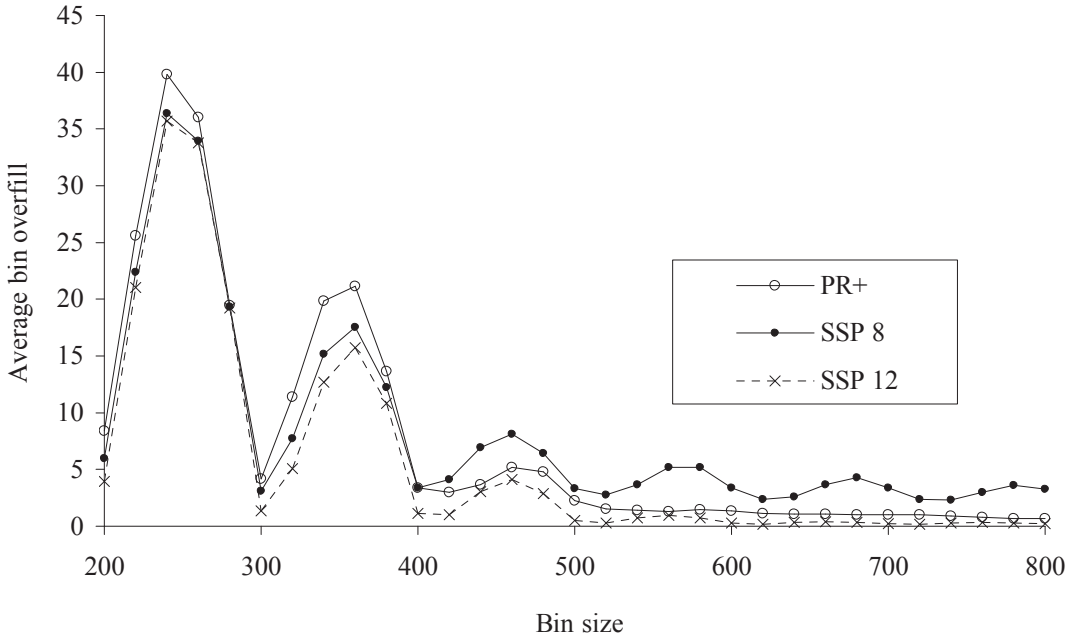


Figure 7.4: Comparison I of the Subset-Sum and Prospect Algorithms for the $N_D(100,15)$ distribution. The bin size is varied.

be fixed at 60% for all cases. Based on this observation, one would prefer the SSP algorithm over PR+, if the mechanics of the machine were not a factor.

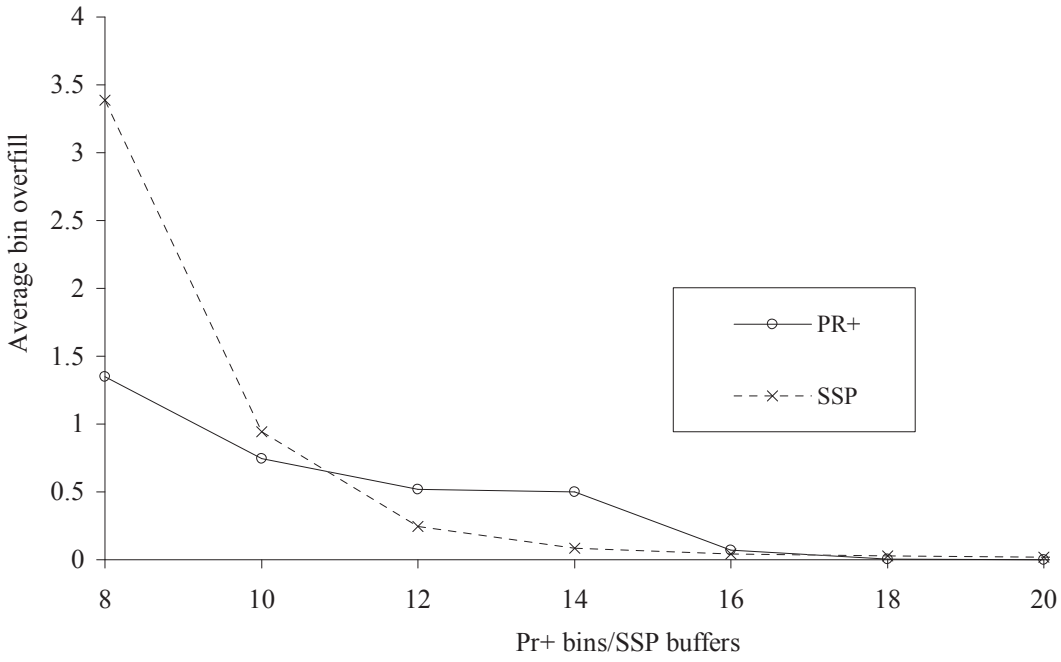
7.4 The Multiple Subset-Sum Problem

This problem corresponds to a grader that can weigh l items in advance on the conveyor belt. This is mechanically simple; one only needs to add an extra conveyor belt between the scale and the bins that is big enough to keep the l known items. Figure 7.6 shows a drawing of such a machine.

The formal definition of the problem is:

Definition 7.2 (l-buffered on-line constraint) The items arrive on-line, and they still have to be allocated in the given order, but now the next l items are known in advance. Note that if $l = 1$, this constraint reduces to the regular on-line constraint definition (see Page 4).

If there is no constraint on l, there obviously are enough known items to fill all



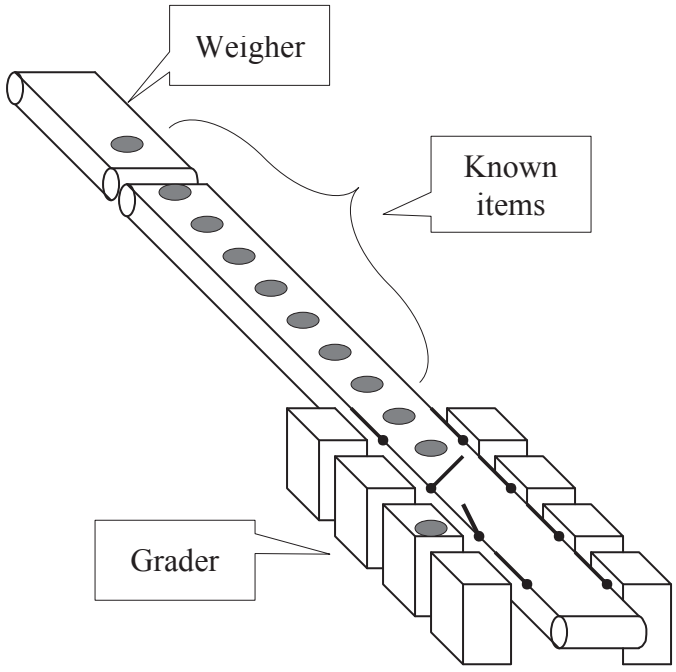


Figure 7.6: Grader with several known items.

active bins, but there is little advantage of having more items than those that fill all the available bins (since it is assumed that all items are allocated to bins in order). The case with no constraint on l is labeled $l = \infty$. In that case, the Bin-Covering assignment can be done by solving a series of Subset-Sum problems, one for each bin to be filled (the bins are filled in order by increasing empty space). Only the assignment for the first item in the list is used; after the first item has been allocated the algorithm is run again to get an allocation for the next item on the list, and so on. The number of items needed is estimated to be just enough to fill all bins plus one extra item. The estimate of the number of needed items can be too low (i.e., all the bins will not be filled), but nothing extra is done in that case; the allocation for the first item is used as is. In the rare case that the next item in line does not get selected for any bin, that item is put in the bin with the largest empty space.

The approximation algorithm described in the previous paragraph is called the *Multiple Subset-Sum* (MSS) algorithm, and it was first put forth by Hjaltason [73]. His studies have suggested that the MSS algorithm achieves better results than the Prospect Algorithm, which is not surprising since now there is much more information available on the incoming items. This approach seems to work well in some cases, as seen in Figure 7.7 ($l = \infty$). The total overfill of the first 100,000 batches for varying bin capacity is plotted. Note that in many instances the overfill is zero, implying that the performance is optimal.

However, the MSS algorithm requires that there be enough items to choose from so that the algorithm can fill all the active bins, but not more. The algorithm will not work unmodified when l is finite, so that it not guaranteed that all unfilled bins can be filled with items whose weight is known. We have modified the MSS algorithm to work with any value of l, and call this new version MSS_l; the original algorithm is then labeled MSS_∞. The trick is to use *imaginary* items based on the item distribution. To generate m imaginary pieces from the underlying distribution, the distribution is

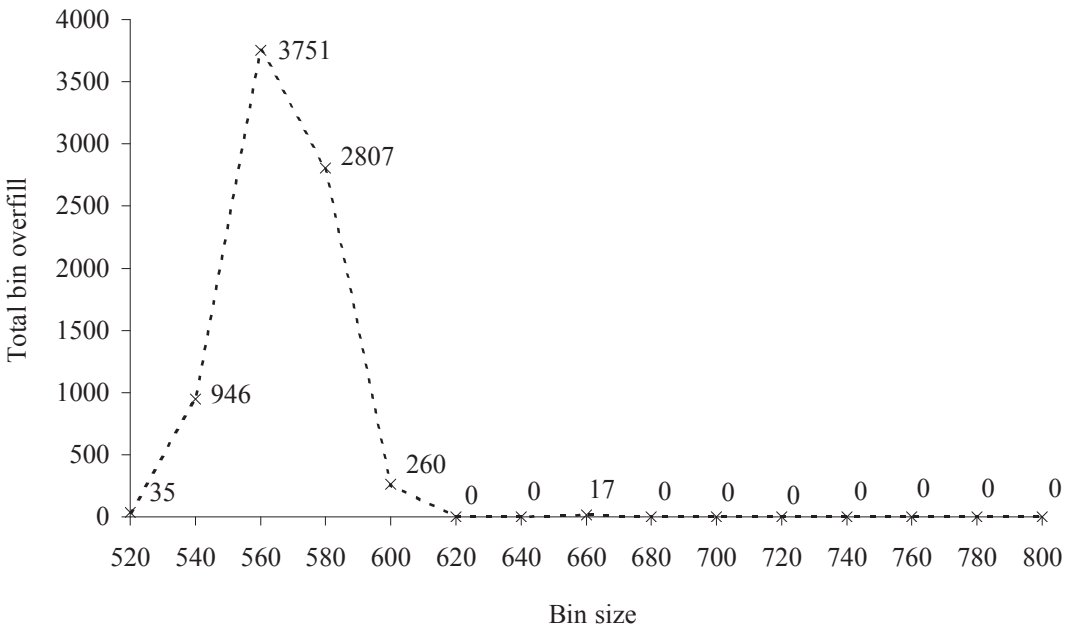


Figure 7.7: Total overfill for the MSS algorithm after 100,000 filled bins. Item distribution is $\mathrm{N}_D(100,15)$ and there are eight active bins.

split into m percentiles and the average size of each percentile calculated. These m averages are the imaginary items used. The number of imaginary items m is chosen so that $m + l - 1$ items fill all the active bins, as with the MSS_∞ algorithm. This new list of $m + l$ real and imaginary items is then run through the MSS algorithm to obtain a decision on the real items.

To compare MSS_l to the Prospect Algorithm PR+, a series of simulations are performed. The number of bins is eight in all of the simulations, the bin size is varied from 200 to 800 in increments of 20, each simulation is run until 10,000 bins have been filled, and the optimal zone parameter for the PR+ algorithm is determined with the Optimal Zone Search Algorithm 3.3 on Page 38. The item distributions $\mathrm{N}_D(100,10)$, $\mathrm{N}_D(100,15)$, $\mathrm{U}_D(75,125)$, and $\mathrm{N}_D(100,20)$ are used. For MSS_l, $l \in \{1, 2, \ldots, 20, \infty\}$, Figures 7.8, 7.10, 7.12, and 7.14 depict the average overweight for each distribution and different values of l compared with PR+.

Figures 7.8, 7.10, 7.12, and 7.14 show that MSS_l performs better as l increases,

as expected, since more information on the real size of the incoming items is known. The value of l where the MSS_l performs better than PR+ depends on the problem. For distributions $U_D(75,125)$ and $N_D(100,20)$, MSS_l outperforms PR+ for $l \geq 4$; for $N_D(100,20)$, $l \geq 5$ is needed; but for $N_D(100,10)$, MSS_l always beats PR+. The reason for this can be seen in Figures 7.9, 7.11, 7.13, and 7.15 where the MSS_1, MSS_∞, and PR+ overfill is shown for all the bin sizes and each distribution. MSS_1 tends to do better than PR+ when the item distribution fits badly with the bin size, and hence all algorithms have large overfill. PR+, however, does better than MSS_1 when overweight is low. The reason for this might be that the PR+ algorithm does not look at the overfill directly, but is trying to maximize the chances of bins ending up within the predefined zone, whereas MSS_l is trying to minimize overfill directly in its bin allocation, and this penalizes the PR+ algorithm when the problem is hard (i.e., high overfill). The $N_D(100,10)$ distribution fits badly with most bin sizes, making MSS_1 do relatively well, whereas the other distributions fit well with the bin sizes, making PR+ perform better there.

The Prospect Algorithm also can be modified to take into account knowledge of the next few item sizes. The simplest solution is to try all placement combinations of the next l items, and select the allocation for the next item from the allocation of all the items that resulted in the best positive change in the Prospect function. This is a brute force method that explodes very quickly in time, with the number of operations growing as k^l, where k is the number of active bins. We compare this modified Prospect Algorithm, $PR+_l$, to MSS_l and the results are shown in Figure 7.16. The Prospect Algorithm dominates, probably because it makes better use of the information given in the distribution than the MSS_l algorithm. However, because of its calculation time explosion, it can only run for $l \leq 5$, whereas the calculation time of MSS_l is almost independent of l (since it takes much less computation to generate the imaginary pieces than to solve the series of Subset-Sum problems in

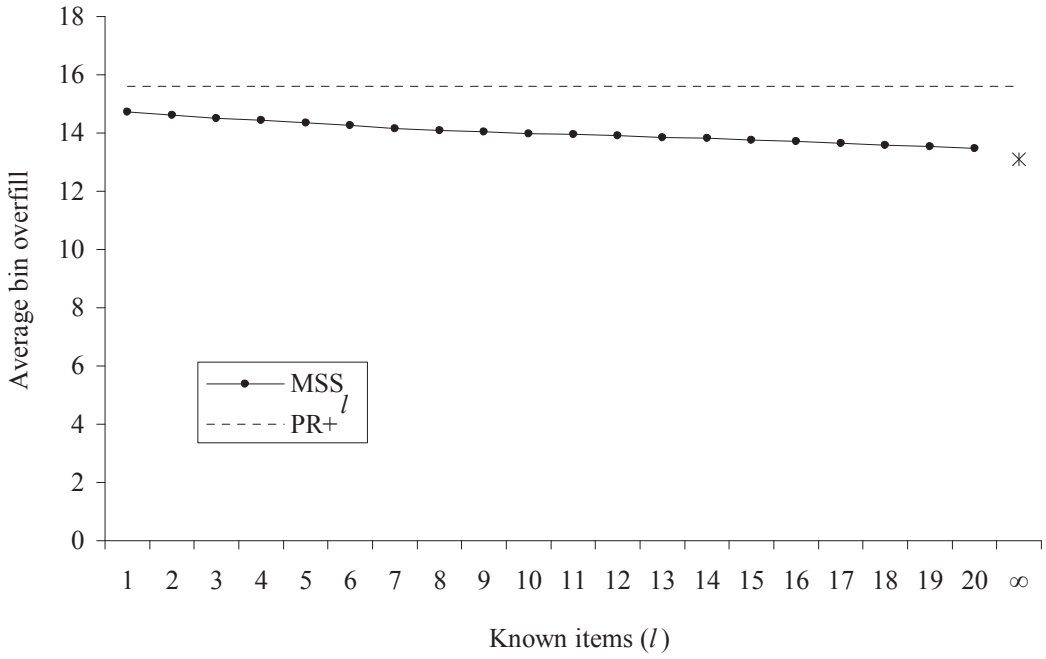


Figure 7.8: Comparison of average overfill for MSS_l and PR+ for the $\mathrm{N}_D(100,10)$ distribution and eight bins.

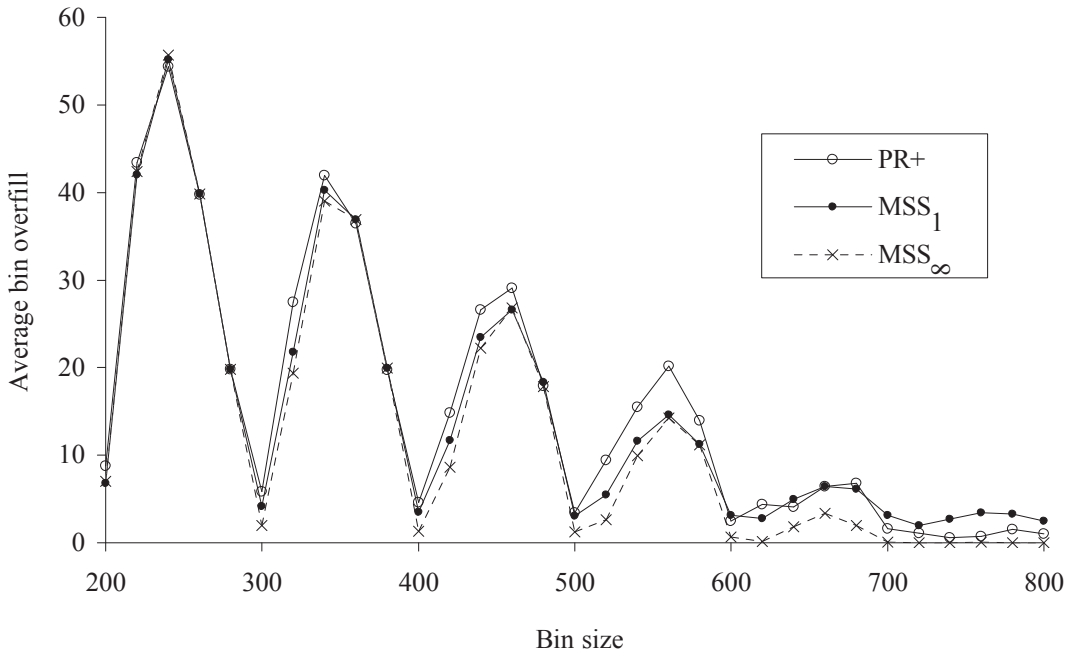


Figure 7.9: Comparison of MSS_1, MSS_∞, and PR+ for the $\mathrm{N}_D(100,10)$ distribution and eight bins.

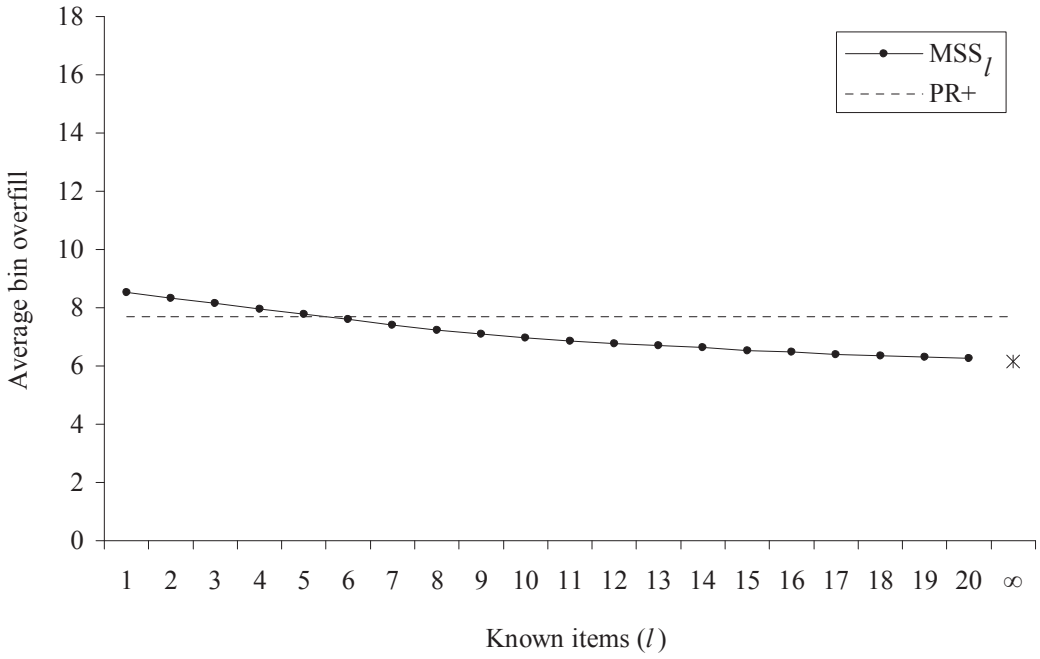


Figure 7.10: Comparison of average overfill for MSS_l and PR+ for the $\mathrm{N}_D(100,15)$ distribution and eight bins.

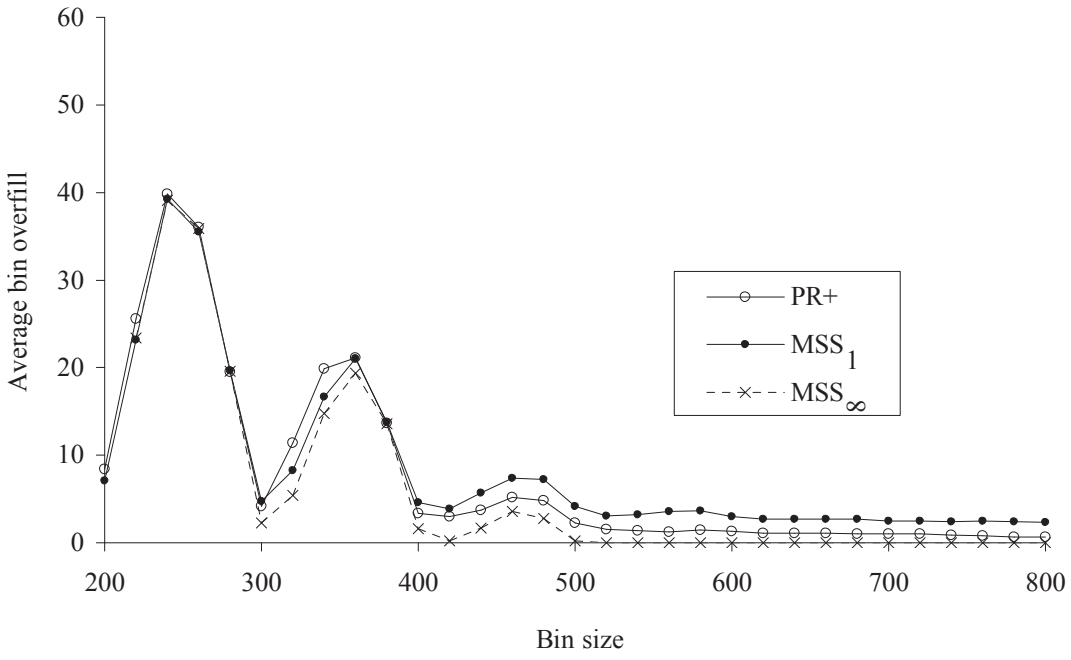


Figure 7.11: Comparison of MSS_1, MSS_∞, and PR+ for the $\mathrm{N}_D(100,15)$ distribution and eight bins.

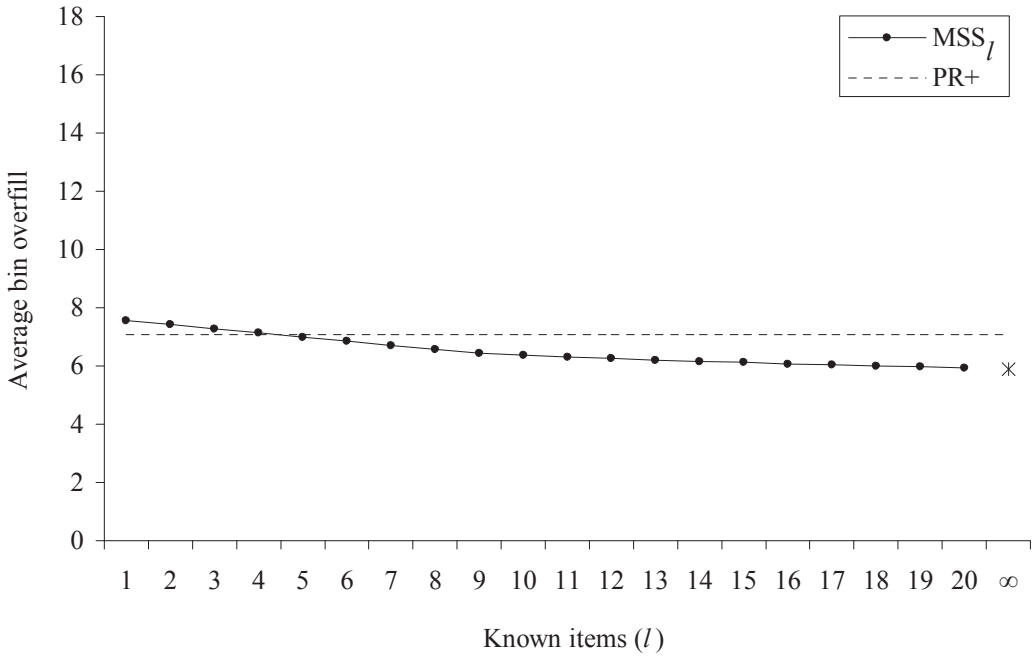


Figure 7.12: Comparison of average overfill for MSS_l and PR+ for the $\mathrm{U}_D(75{,}125)$ distribution and eight bins.

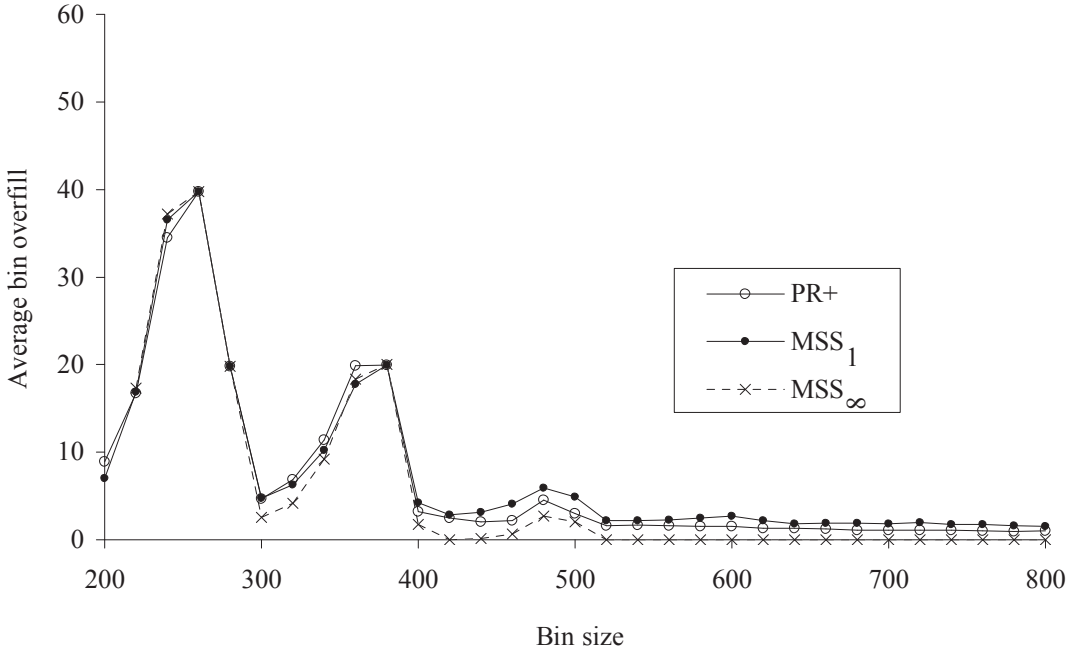


Figure 7.13: Comparison of MSS_1, MSS_∞, and PR+ for the $\mathrm{U}_D(75{,}125)$ distribution and eight bins.

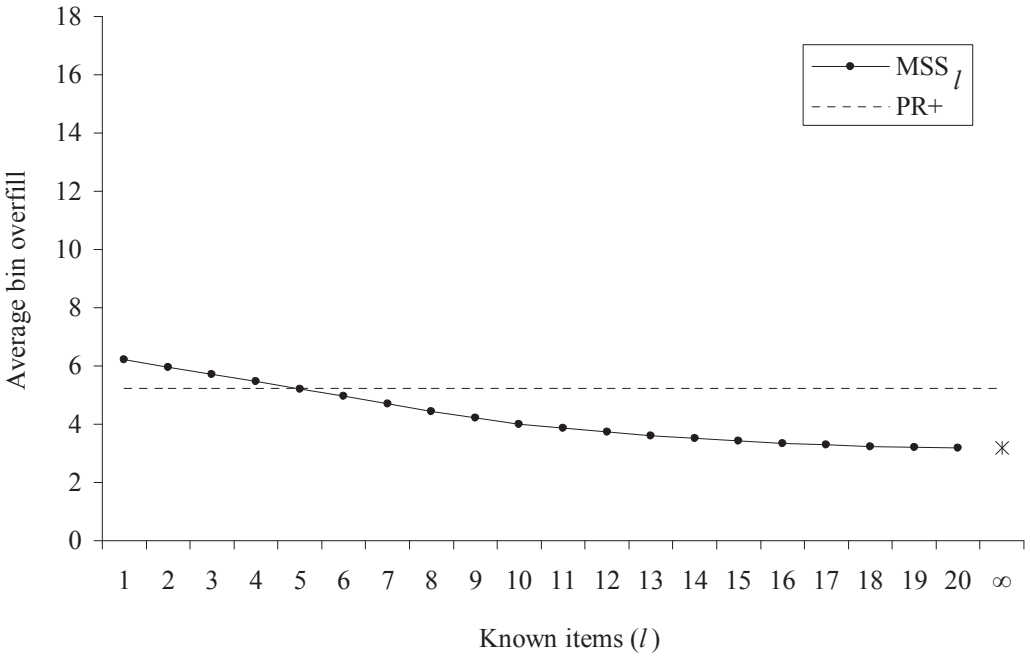


Figure 7.14: Comparison of average overfill for MSS_l and PR+ for the $N_D(100,20)$ distribution and eight bins.

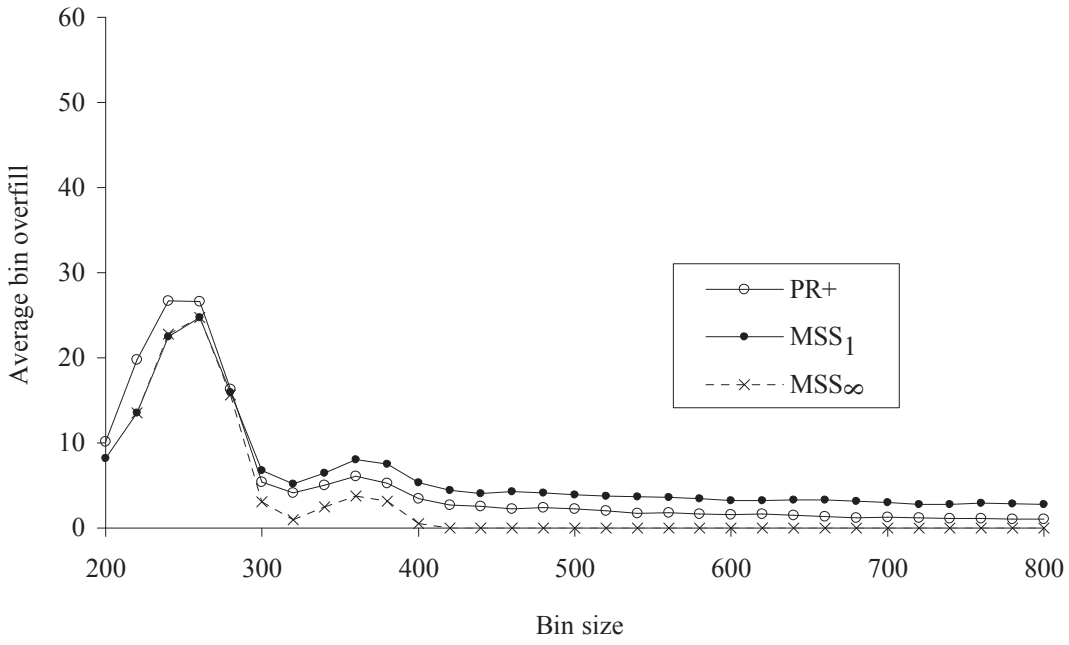


Figure 7.15: Comparison of MSS_1, MSS_∞, and PR+ for the $N_D(100,20)$ distribution and eight bins.

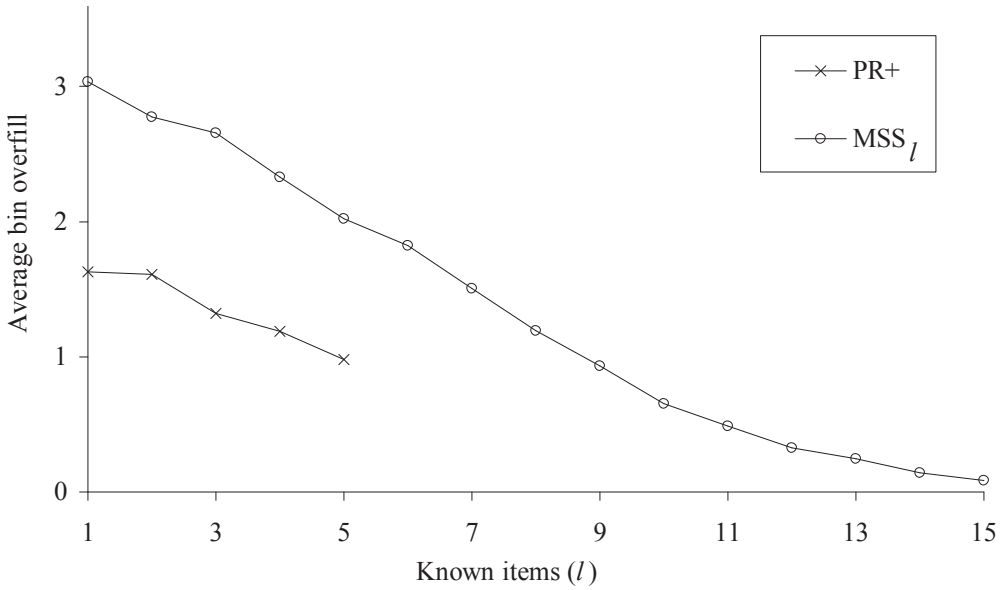


Figure 7.16: Comparison of the MSS_l and PR_l algorithms for the $\mathrm{N}_D(100,15)$ distribution, eight active bins, and bin size 600.

MSS_l). There is of course a possibility that a practical way of running $\mathrm{PR}+_l$ for $l > 5$ can be developed, but it has not been done to date.

CHAPTER 8

PRACTICAL APPLICATION OF THE PROSPECT ALGORITHM

So far in the thesis it has been assumed that the distribution of items is known, but in real-world applications that is not the case, so the distribution has to be estimated on-line, and the Prospect function recalculated constantly. And what is more, the distributions that are encountered can be very different. These issues are probed in this chapter. First, in Section 8.1, the main types of item distributions that are encountered in the real-world are described. Second, in Section 8.2, the concept of item *doubling* is introduced, and a new algorithm to monitor item doubling is described and tested. In Section 8.3, we describe how the item distribution is estimated on-line, and how the Prospect function can be kept up-to-dat. In Section 8.4, parameter choices for on-line item distribution estimation and the Prospect function calculation are examined, and the different ways to calculate the Prospect function are compared using simulation. Finally, in Section 8.5, different versions of the Prospect Algorithm set forth in this thesis are compared from a practical point of view.

8.1 Real Product Distributions

As described in the introduction Chapter 1, Marel graders are used for grading and packing many different types of food products, almost always some sort of meat or fish. The underlying item distribution is usually close to being a normal distribution, but the items are commonly graded by weight as well as packed, and this can have a significant impact on the distribution of items that go into each pack. Figure 8.1 shows a hypothetical case where items are graded into three weight grades, A, B, and

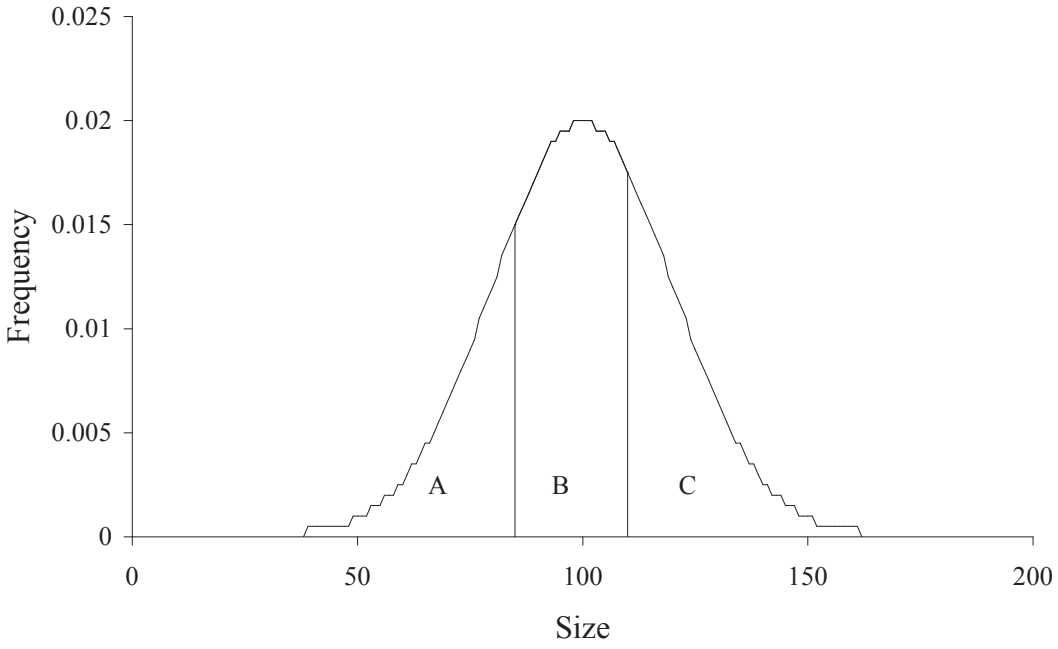


Figure 8.1: $N_D(100,20)$ distribution split into 3 grades, A= $(0, 85]$, B= $(85, 110)$ and C= $[110, \infty)$.

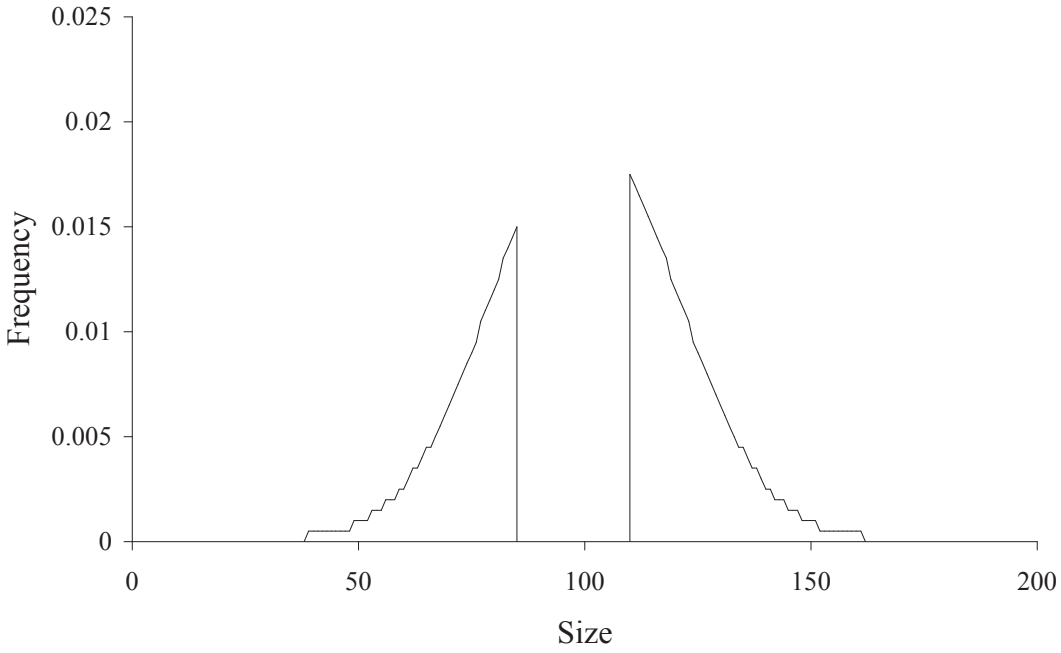


Figure 8.2: $N_D(100,20)$ distribution with grade B= $(85, 110)$ removed.

C. Grades A and C are heavily skewed since they are from the tails of the distribution, but Grade B is more symmetrical. Figure 8.2 shows a more-extreme case, assuming that grade B is filtered out into other processing and the leftover grades A and C (A+C) are packed together. This distribution has two peaks, which makes it look totally different from the other distributions, and might suggest that the twin peaks will have the greatest effect on performance. As shown below, however, this is not the case.

To show how grading such as described above affects bin covering performance, simulations using the Prospect+ Algorithm described in Section 5.5 were carried out using data from the $N_D(100,20)$ distribution, graded into the grades A, B, C, and A + C. The number of bins is eight in all of the simulations, the bin size is varied from 200 to 800 in increments of 10, each simulation is run until 20,000 bins have been filled, and the optimal zone parameter is determined with the Optimal Zone Search Algorithm 3.3 on Page 38. Figures 8.3 to 8.6 show how bin covering performance for the different grades compares to using the entire distribution. The important factor affecting the performance is the relative variance (σ/μ). Grading the distribution changes relative variance significantly as depicted in Table 8.1, and this effect can be seen in the performance figures.

More specifically, lowering the relative variance makes the performance more dependent on the bin size, since as the relative variance decreases, most items become similar to each other in weight, providing fewer possibilities for the bin covering algorithm to mix items of different weights to achieve a filled bin close to the target size. Grades B and C (Figures 8.4 and 8.5, respectively) have the lowest relative variance; so their performance varies most with bin size. Grade A (Figure 8.3) is slightly better, even though it is much more sensitive to bin size than the entire distribution. Grade A + C (Figure 8.6) has by far the highest relative variance, higher than the original distribution since items around the mean have been removed. Bin covering

Table 8.1: Relative variance (σ/μ) for N_D(100,20) and its grades.

Items	μ	σ	σ/μ
N_D(100,20)	100.0	20.0	20.0%
Grade A	73.8	9.6	13.0%
Grade B	97.8	6.8	6.9%
Grade C	122.5	10.4	8.5%
Grade A + C	101.8	26.1	25.7%

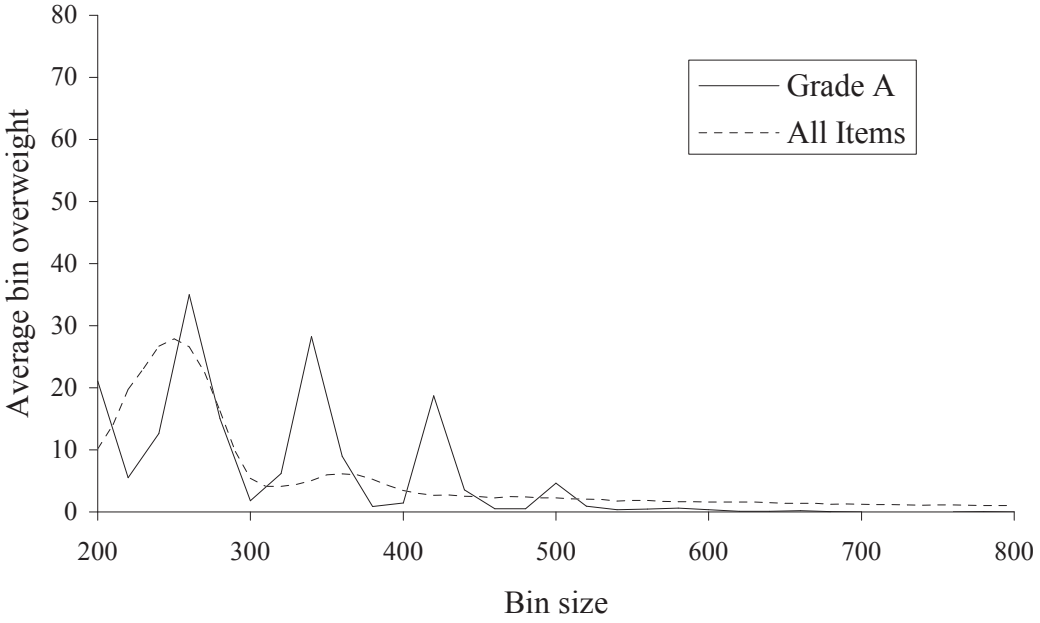


Figure 8.3: Performance of the Prospect+ Algorithm for grade A compared to all items of the N_D(100,20) distribution.

performance for grade A + C is therefore close to the performance of the original distribution, sometimes below, sometimes above, and often about equal. This analysis demonstrates the tradeoffs that occur when bounds are set on the weight of individual items while one attempts to make all filled bins similar in weight.

8.2 Double Effect

Another process-dependent distribution effect is what can be called the *double effect*, where two (sometimes even more) pieces are loaded together onto the grader and are weighed and routed together after that. The product that is being processed is

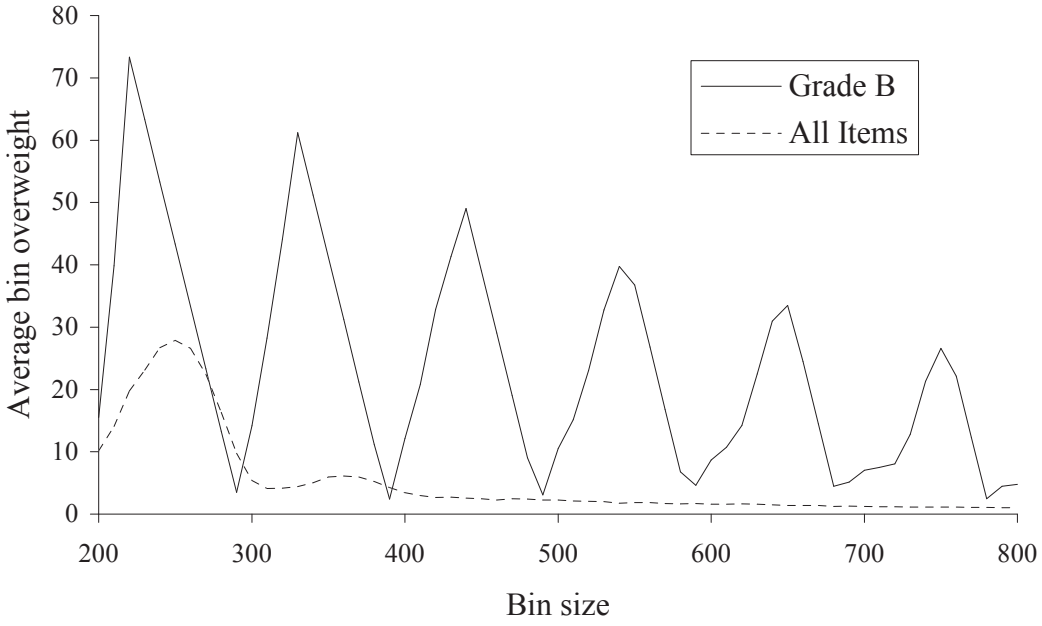


Figure 8.4: Performance of the Prospect+ Algorithm for grade B compared to all items of the $N_D(100,20)$ distribution.

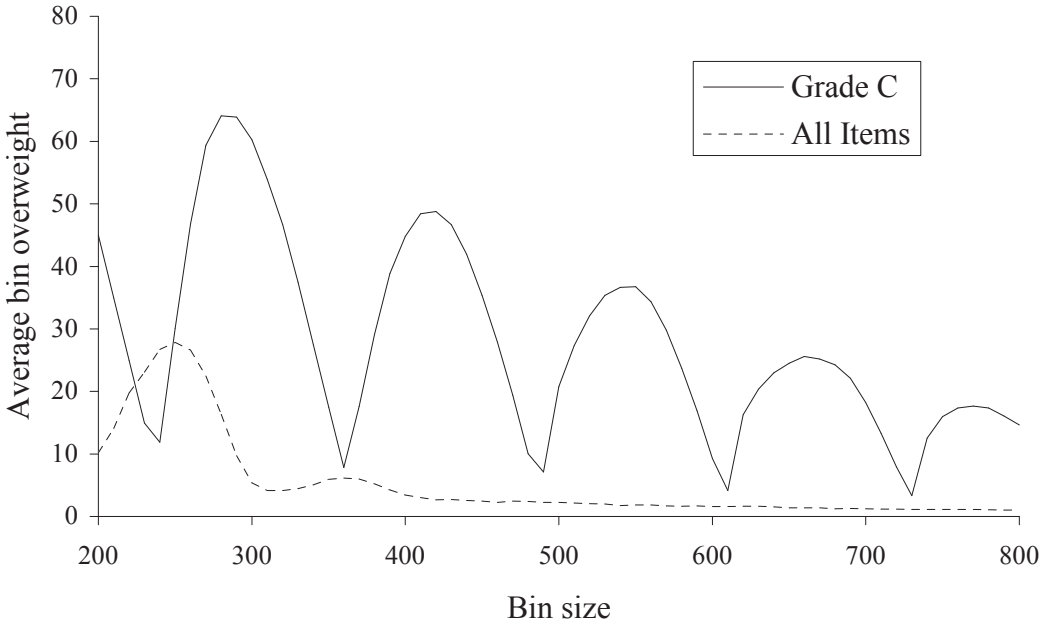


Figure 8.5: Performance of the Prospect+ Algorithm for grade C compared to all items of the $N_D(100,20)$ distribution.

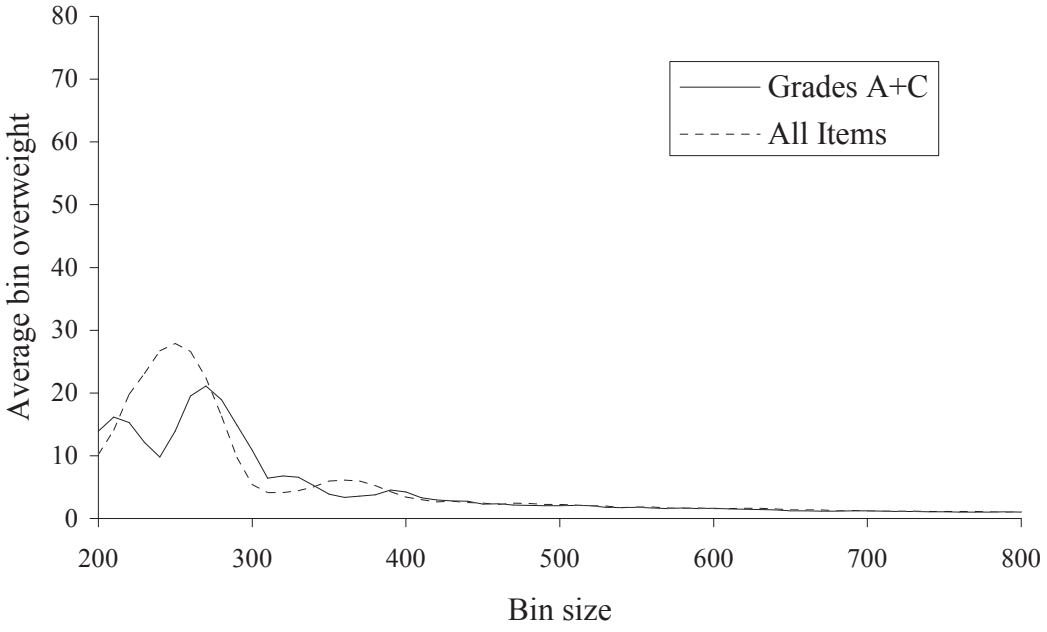


Figure 8.6: Performance of the Prospect+ Algorithm for grades A + C compared to all items of the $N_D(100,20)$ distribution.

frequently loaded into the system via slotted in-feed conveyor belts, and if multiple pieces end up in the same slot, they will travel together through the system. This shows up as a second hump on the measured item distribution, as seen in Figures 8.7 and 8.8. Figure 8.7 shows a hypothetical normal distribution that has 10% of items doubled. Figure 8.8 shows real production data — chicken wings — of which approximately 5% are doubled. The data was logged in 2004 on a Marel grader at the Pilgrim's Pride poultry plant in Lufkin, Texas.

Doubling like this affects performance directly; and to illustrate this phenomenon, we ran a simulation using the Prospect+ Algorithm and the $N_D(100,15)$ item distribution. The number of bins is eight in all of the simulations, the bin size is 300, 400, or 500, each simulation is run until 10,000 bins have been filled, and the optimal zone parameter is determined via the Optimal Zone Search Algorithm 3.3. The doubling ratio is varied in the simulations from 0% to 25% in 5% increments. Figure 8.9 shows the resulting average bin overfill as a function of doubling frequency. The smaller the

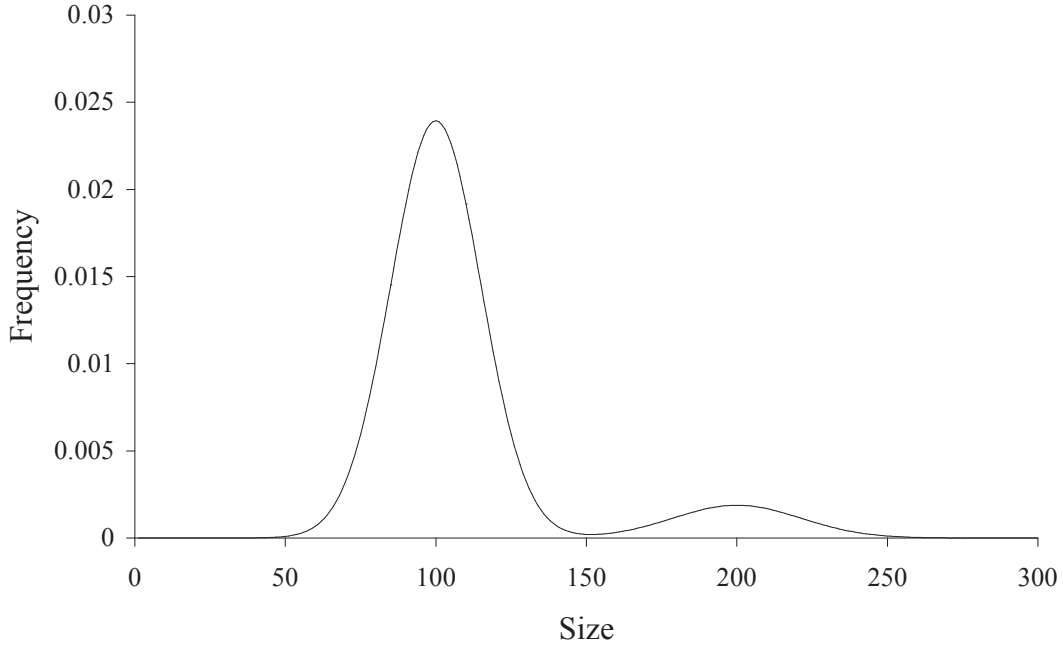


Figure 8.7: Normal $(100, 15)$ distribution with 10% of items doubled.

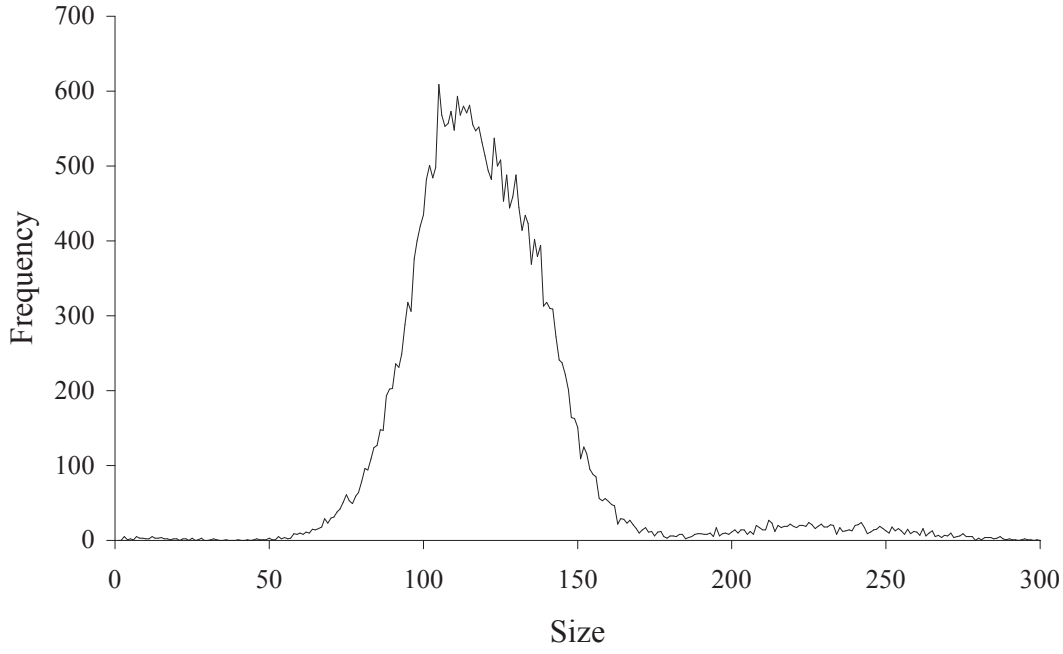


Figure 8.8: Real item distribution (chicken wings, approximately 5% are doubled).

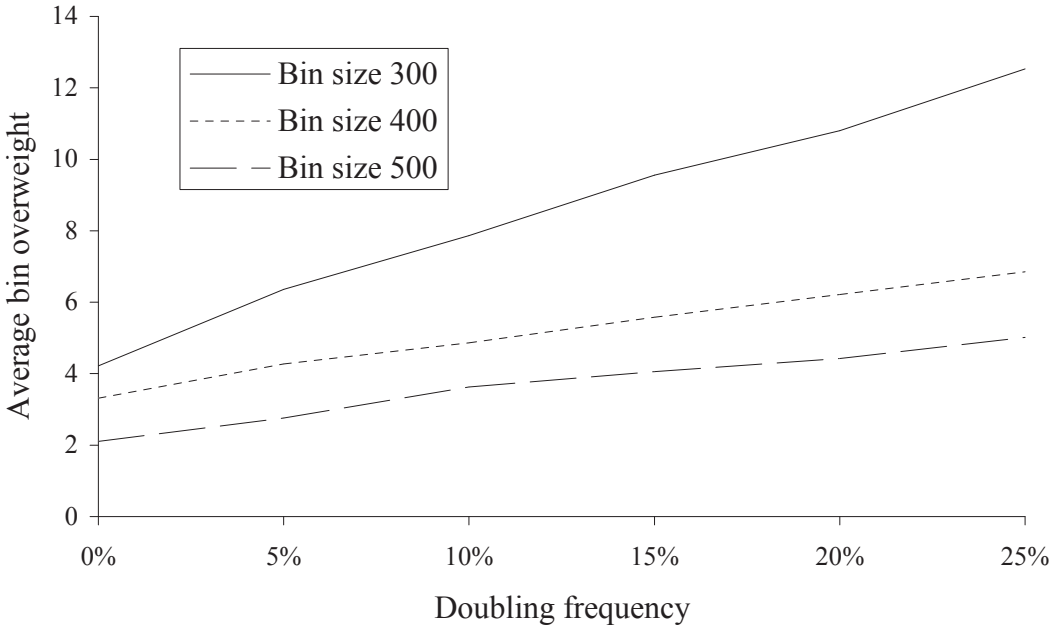


Figure 8.9: Effect of increased doubling on bin overfill. Data is from the $N_D(100,15)$ distribution and there are eight bins.

bin is, the greater is the effect of item doubling on performance.

Detecting the doubling frequency of pieces is also an interesting problem on its own for two other reasons:

1. A high doubled ratio indicates problems in the loading process. There could be a mechanical problem, or the workers loading the machine may not be careful enough to put one item per each loading slot, making some empty and some with doubles or triples.

2. Estimating the item average is useful for process monitoring, and the simplest measure of item average, the mean, is sensitive to outliers like the doubled items.

In particular, doubling makes the calculated sample mean, $\hat{\mu}$, higher than the true underlying mean μ of the item distribution. There can of course be even more than two items per slot, provided that there is room in the slot for such a number of items. Denoting $f(c)$ as the probability mass function of the distribution of the number c of

items per slot, $\hat{\mu}$ is going to be:

$$\hat{\mu} = \mu \sum_{c=1}^{\infty} f(c) \tag{8.1}$$

(assuming that doubling frequency is independent of the item size). Therefore if one has an estimate of the underlying mean, the average number of items per slot can be estimated as $\hat{\mu}/\mu$. This estimate of the average number of items per slot can be used as an indicator of doubling in process monitoring. To estimate μ, one possibility is to use the *mode* M. A well-known algorithm for estimating the mode in a finite sample, the *half-range mode* (HRM), is set forth by Bickel [17]. To explain how HRM works, it helps first to look at a method for calculating the mode of a single-modal continuous distribution with cumulative distribution function $F(\cdot)$. Define the *modal interval* of width υ as the interval $[\tilde{M}(\upsilon) - \upsilon/2, \tilde{M}(\upsilon) + \upsilon/2]$, where the midpoint, $\tilde{M}(\upsilon)$, is a value that maximizes $F(\tilde{M}(\upsilon) + \upsilon/2) - F(\tilde{M}(\upsilon) - \upsilon/2)$. As the width decreases, the midpoint of the modal interval approximates the mode, or:

$$\lim_{\upsilon \to 0} \tilde{M}(\upsilon) = M. \tag{8.2}$$

This provides an idea for estimating the mode for a finite sample. First a discrete version of a modal interval, *discrete modal interval*, is defined with the following algorithm:

Algorithm 8.1 (Discrete Modal Interval) The input is an ascending ordered list of sample values $\mathbf{v} = \{x_1, x_2, \ldots, x_n\}$, and the desired interval width υ. The output is a selection from it, denoted as the discrete modal interval with k items, $\mathbf{M_d}(\upsilon) = \{x_j, x_{j+1}, \ldots, x_{j+k-1}\}$, and is calculated with the following algorithm.

Step 1: Loop over all possible intervals from $\mathbf{v}$, and collect in a list L all intervals whose range is less than or equal to υ and number of items, n', is as large as possible. If there is exactly one such interval, return it as the discrete modal interval $\mathbf{M_d}(\upsilon)$, $k = n'$; else go to Step 2.

Step 2: Remove all intervals from the list L whose range is greater than the smallest range of all the intervals. If there is exactly one interval left in L, return it as the discrete modal interval $\mathbf{M_d}(v)$, $k = n'$; else go to Step 3.

Step 3: Loop over all the intervals in L, and find the minimum and maximum items in them. Return as the discrete modal interval $\mathbf{M_d}(v)$ all k items that are within those minimum and maximum values. In this case $k > n'$.

Using this discrete modal interval, the HRM Algorithm by Bickel [17] can then be described as a discrete version of Equation (8.2):

Algorithm 8.2 (HRM) Start with an ascending ordered list of values $\{x_1, x_2, \ldots, x_n\}$, so that $x_1 \leq x_2 \leq \cdots \leq x_n$.

Step 1: Let $\mathbf{v} = \{v_1, v_2, \ldots, v_n\}$ denote the vector of n current values in the algorithm. Initialize with the original list of values, or $\mathbf{v} = \{x_1, x_2, \ldots, x_n\}$. Go to Step 2.

Step 2: If $\mathbf{v}$ contains only one or two values, then return the mean of those values as the mode, $\tilde{m}$, and stop; otherwise, proceed to Step 3.

Step 3: Set the interval width w to half the range of $\mathbf{v}$, i.e., let $w = \frac{1}{2}(v_n - v_1)$. Calculate the discrete modal interval $\mathbf{v}' = \mathbf{M_d}(w)$, and set n' as the number of values in it. If $n' < n$, then set $\mathbf{v} = \mathbf{v}'$, $n = n'$, and go to Step 2. Else go to Step 4.

Step 4: If $v_2 - v_1 < v_{n'} - v_{n'-1}$ then drop $v_{n'}$ from $\mathbf{v}'$, set $\mathbf{v} = \mathbf{v}'$, $n = n' - 1$, and go to Step 2. Else go to Step 5.

Step 5: If $v_2 - v_1 > v_{n'} - v_{n'-1}$ then drop v_1 from $\mathbf{v}'$, set $\mathbf{v} = \mathbf{v}'$, $n = n' - 1$, and go to Step 2. Else go to Step 6.

Step 6: If $v_2 - v_1 = v_{n'} - v_{n'-1}$ then drop v_1 and $v_{n'}$ from $\mathbf{v}'$, set $\mathbf{v} = \mathbf{v}'$, $n = n' - 2$, and go to Step 2.

There are, however, two significant problems with using the mode as an estimator for μ. First, HRM estimation of the mode is quite noisy (as seen in Figures 8.15 to 8.18 below), and second, the mode itself is not an unbiased estimator of the mean unless the distribution is symmetric. To tackle this problem, we suggest a two-step approach and a new algorithm is proposed. This new algorithm shows promising results both on simulated and real data.

The idea behind the algorithm is to first estimate the location of the big *hump* in the distribution, as seen in Figures 8.7 and 8.8, and then calculate the average of the items within the distribution hump and use that as an estimate of μ. To estimate the location of the hump, the discrete modal interval from Algorithm 8.1 is found for a "suitable value" of υ. Figure 8.10 shows this graphically, using the same distribution as in Figure 8.7 and a modal interval width $\upsilon = 50$. In the figure, the interval captures about 81.4% of the distribution, and the average of those items will be close to the underlying average μ. Setting the modal interval width υ is critical, and experiments indicate that defining this as a fixed ratio of the mode as calculated by the HRM Algorithm works well. A parameter ρ is defined and then the modal interval size (υ) is calculated as:

$$\upsilon = \rho\tilde{m}, \tag{8.3}$$

where $\tilde{m}$ is the mode as calculated by the Bickel Algorithm 8.2. Good choices for values of ρ are examined later in this section. The new proposed algorithm is called the *Auto Range Mean (Auto RM)* Algorithm, and is more formally described as follows:

Algorithm 8.3 (Auto Range Mean) The input to the algorithm is the range mode ratio parameter ρ, and a list of n items $\mathbf{v} = \{x_1, x_2, \ldots, x_n\}$ that are ordered in ascending order. The output of the algorithm is denoted as $\tilde{\mu}$.

Step 1: Calculate the mode $\tilde{m}$ according to HRM Algorithm 8.2, and set the modal interval size $\upsilon = \rho\tilde{m}$. Go to Step 2.

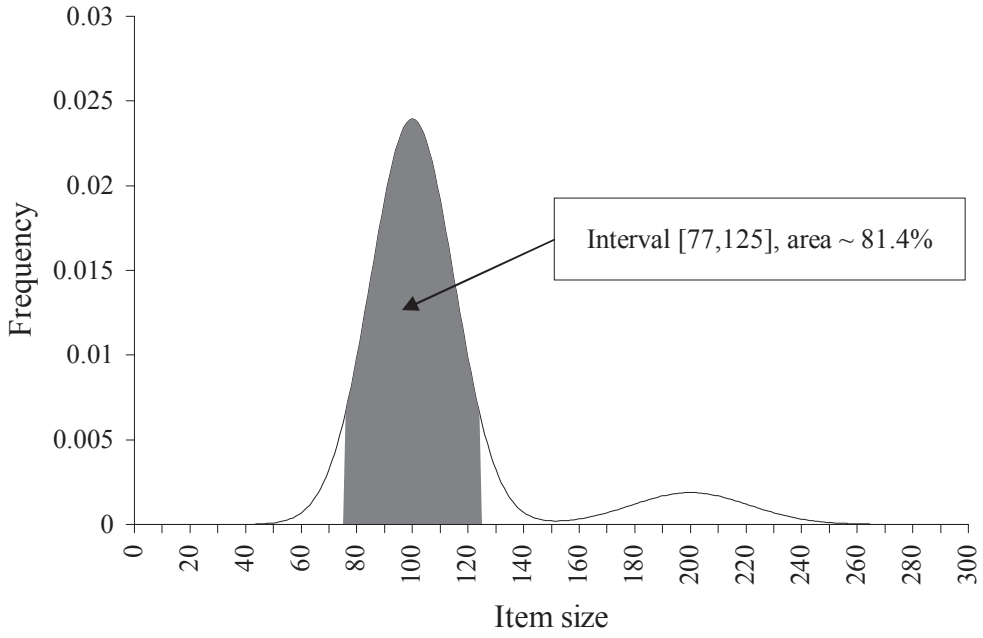


Figure 8.10: Modal interval of width $w = 50$, interval is [75,125]. Distribution is N(100,15) with 10% of items doubled.

Step 2: Sort the items by size. The order can be either descending or ascending. Find the discrete modal interval of width υ using Algorithm 8.1, and calculate the average of those items. Return that average as $\tilde{\mu}$.

As an example of how Auto RM works, we use the first 1000 items of the real product distribution from Figure 8.8 that was logged in the Pilgrim's Pride poultry plant in Lufkin, Texas. The mode according to Bickel's algorithm is 110.3, and ρ is 75%, so the modal interval used is $\upsilon = 82.7$. Figure 8.11 shows the 1000 items sorted by size, and the items included in the Auto RM interval, called the *range mean interval*, whose average is used to estimate $\tilde{\mu} = 118.1$. Figure 8.12 shows the histogram of items, with the items included in the range mean estimator shaded dark. From a visual inspection of the graph, one would deduce that the all the items between 65 and 175 represent single items, and the average of those items is approximately 117.8 which is quite close to the Auto RM estimate.

The main parameter of the Auto RM Algorithm, ρ, needs to be set in a balanced

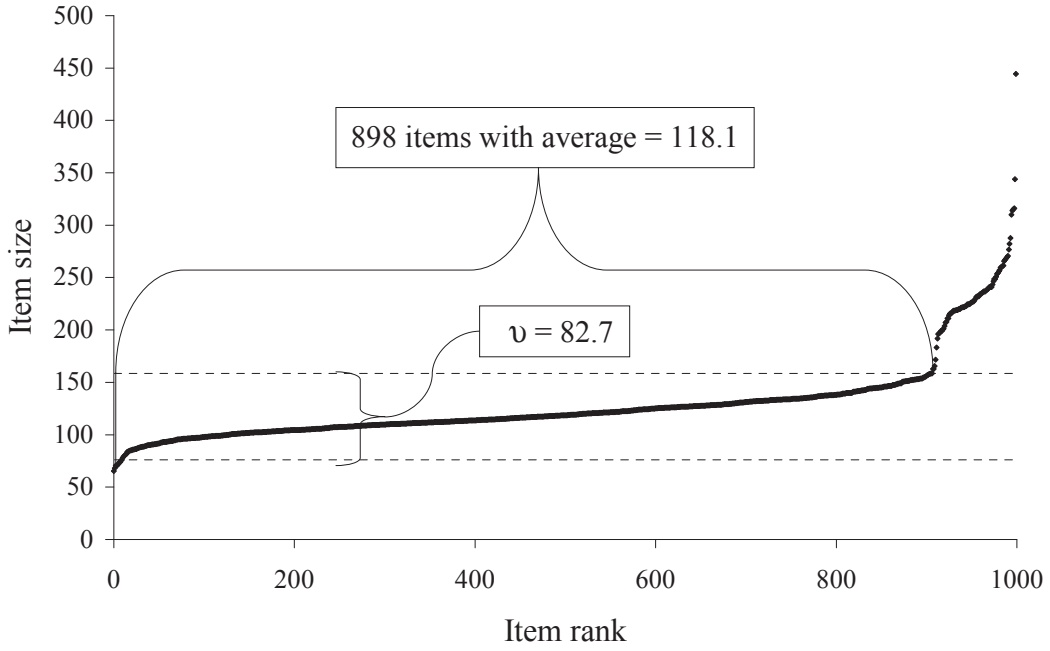


Figure 8.11: Graphical description I of the Auto RM Algorithm. In this case 898 of 1000 items fall within the $v = 82.7$ band, and the average of those items is 118.1.

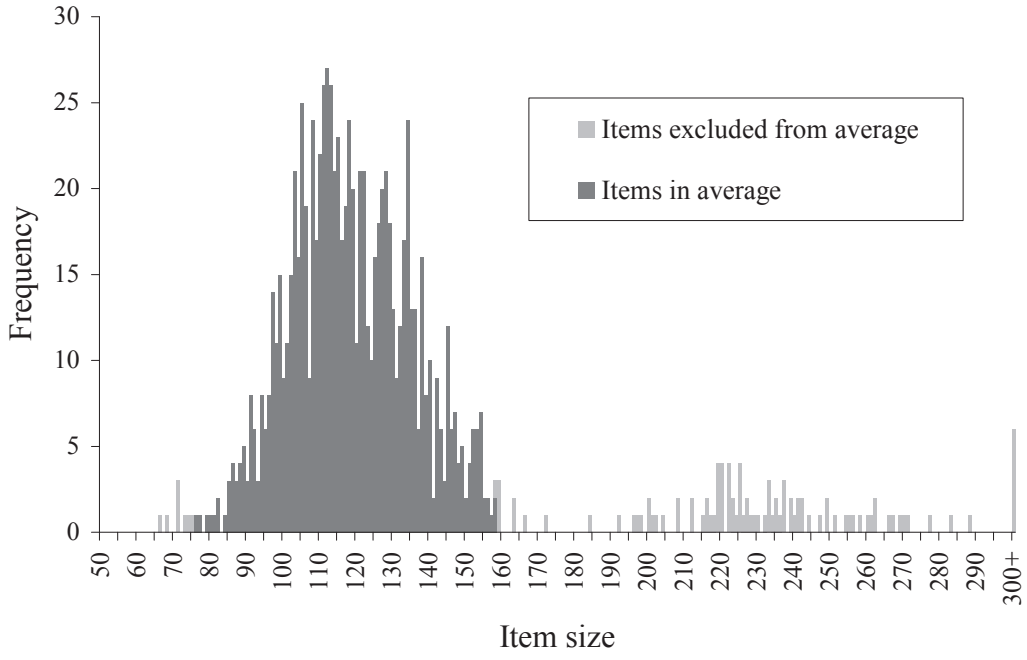


Figure 8.12: Graphical description II of the Auto RM Algorithm. The darker area in the item histogram denotes items included in the range mean interval.

fashion — if it is set too high, it starts to include too many doubled items in the Range Mean, causing the estimator μ to be biased; and if it is set too low, it will become more sensitive to random fluctuations since it will include fewer items in the range mean interval. The length of the FIFO queue of last items weighed (used to estimate the item distribution) is also a parameter and needs to be set appropriately. If it is set too low, the estimation will become inaccurate; and if it is too large, the algorithm might become too slow to track changes in the process.

To test the Auto RM Algorithm, we performed a series of simulations. In all simulations, the items are drawn from a normal distribution of fixed mean $\mu = 100$. For each filled FIFO queue, all necessary statistics were logged so that both the average bias and standard deviation of the item average estimates could be calculated. The simulations were carried out for varying values of item standard deviation σ and doubling frequency d. Standard deviation was varied from 5% to 25% of item average in increments of 5% (5 different values), and doubling frequency was varied from 0% to 30% in increments of 5% (7 different values). These values were chosen based on experience from the food industry and should represent a plausible range of scenarios that might be encountered in practice. The total number of simulation scenarios run for each set of parameters was therefore $5 \times 7 = 35$. The ρ parameter was varied from 50% to 100% in increments of 5% (11 different values), and the FIFO lengths of 250, 500, and 1000 tested (3 different values). The total number of parameter sets was therefore $11 \times 3 = 33$, so the total number of simulation scenarios run was $35 \times 33 = 1155$. For each scenario, 1000 replications of the FIFO queue (for a total of 250,000 to 1,000,000 simulated items) were used to calculate the following two statistics concerning its performance:

1. Relative standard deviation, defined as the standard deviation of the estimator of μ divided by the true item average of the simulation (i.e., the average of the actual items generated in the simulation).

2. Bias, defined as the difference in the estimate from the true item average of the simulation. Positive bias means that the estimate is higher than the true item average.

Figures 8.13 and 8.14 show the average relative standard deviation and average bias plotted, respectively, for all parameter combinations; in particular, the figures depict the averages of all 35 simulations for each parameter set. The average relative standard deviation decreases while the value of ρ increases up to 85–90%, but then the standard deviation begins to ascend. Hence, with respect to the average relative standard deviation, the optimal value of ρ is between 85% and 90%. The average bias, however, increases with an increasing value of ρ. The bias increase is slow at first as ρ grows, but then increases more quickly, especially when ρ passes 75%. The conclusion is that a balance must be found when the value of ρ is chosen. In the rest of the simulations, we select $\rho = 70\%$, since then the bias is quite low (less than 0.15%) but the relative standard deviation is reasonably close to the optimum (within 8% of the average optimum values for all three FIFO sizes).

Figures 8.15 through 8.18 show a comparison of the performance of the sample average, HRM mode, and Auto RM for a symmetric distribution and a skewed distribution. The simulations were performed in a similar fashion as those in Figures 8.13 and 8.14. The symmetric distribution is normal with $\mu = 100$ and $\sigma = 15$, and the skewed data is generated by filtering out items below 90 from the same symmetric distribution. The FIFO queue size is 500 items, and $\rho = 70\%$. The number of doubles increases from 0 to 50% in 5% increments.

Figures 8.15 and 8.17 clearly show how the Auto RM estimator has a lower standard deviation than the HRM mode, and lower than the sample average as well, until doubling approaches 50%. Figure 8.16 shows the bias in the item average estimations for the symmetric distribution. We see that the bias in the sample average estimates

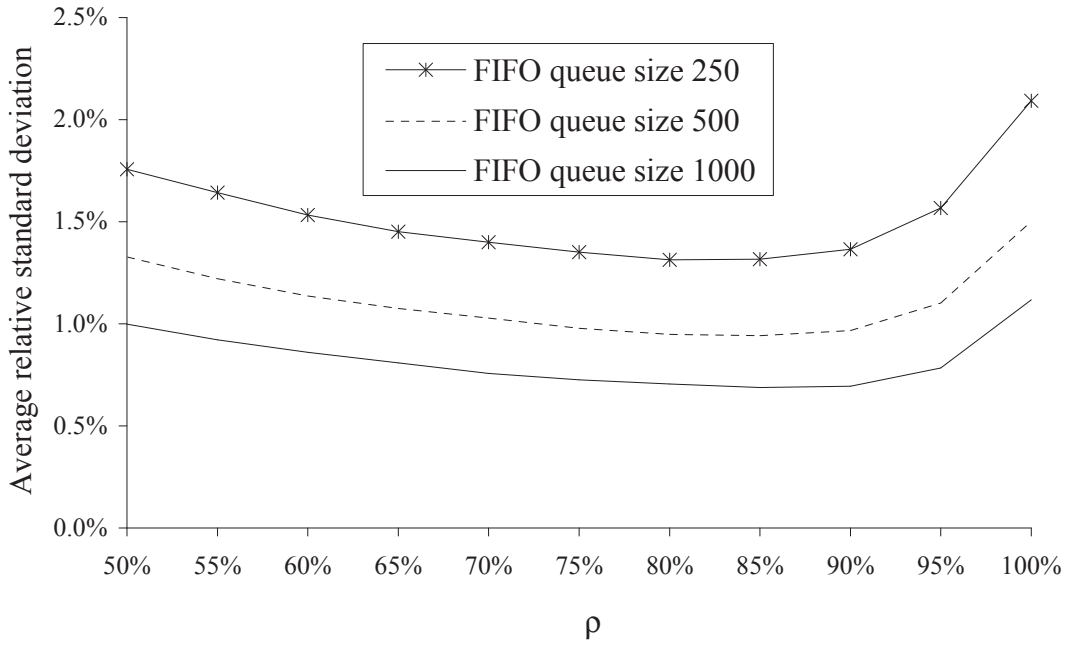


Figure 8.13: Average relative standard deviation of the item mean estimator as a function of ρ for different FIFO queue lengths.

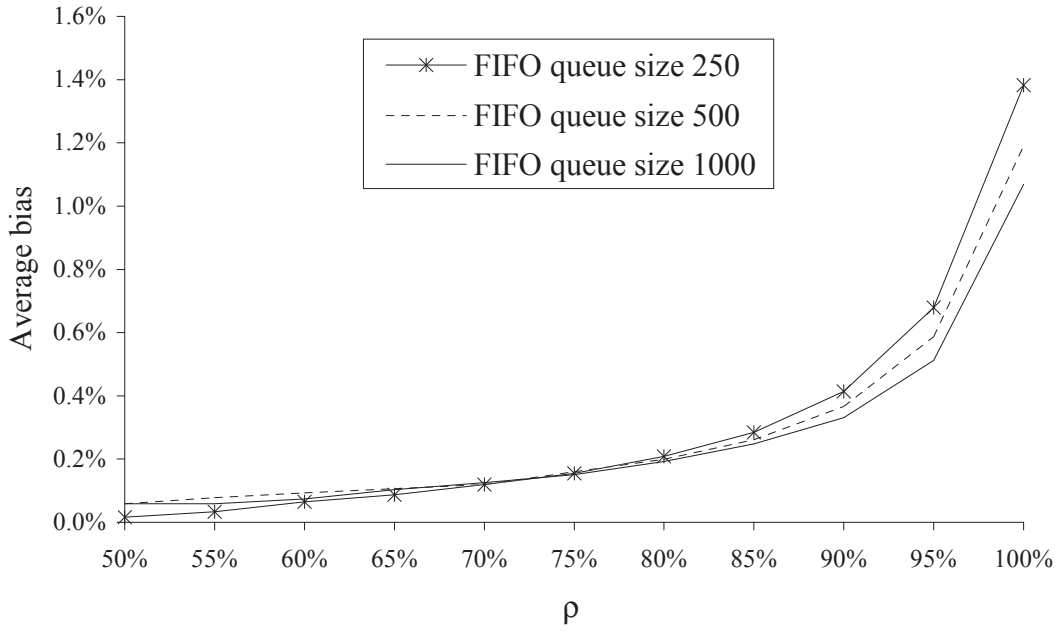


Figure 8.14: Average absolute bias of the item mean estimator as a function of ρ for different FIFO queue lengths.

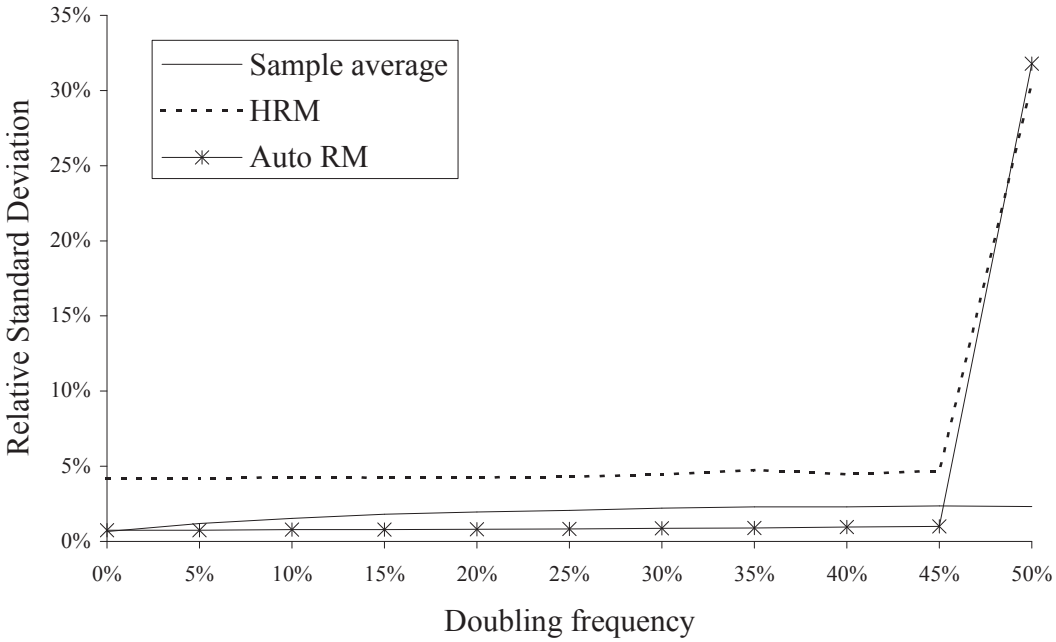


Figure 8.15: Standard deviation comparison of item average estimators for symmetric data.

increases linearly with the doubling percentage; but the other estimators are approximately unbiased until the doubling approaches 50%. Figure 8.18 shows the bias in the item average estimates for the skewed distribution; and now only the Auto RM estimator is approximately unbiased if doubling is less than 50%. Bias of the sample average again increases linearly with doubling percentage, but the HRM mode is also biased. HRM bias is approximately -6%, which is expected, since the true mean is approximately 106.4, while the true mode is still 100. Doubling does not affect the bias of the Auto RM and HRM mode estimates until it reaches 50%, as in the symmetric case. The increase in standard deviation and bias encountered when doubling approaches 50% is not surprising, since in that case there are as many items in the second hump of the distribution as in the first one.

Finally, we test the Auto RM Algorithm on real-world data. The data used is the same as in Figure 8.8. This data for chicken wings was logged on a Marel grader at the Pilgrim's Pride poultry plant in Lufkin, Texas. Logging was carried out on

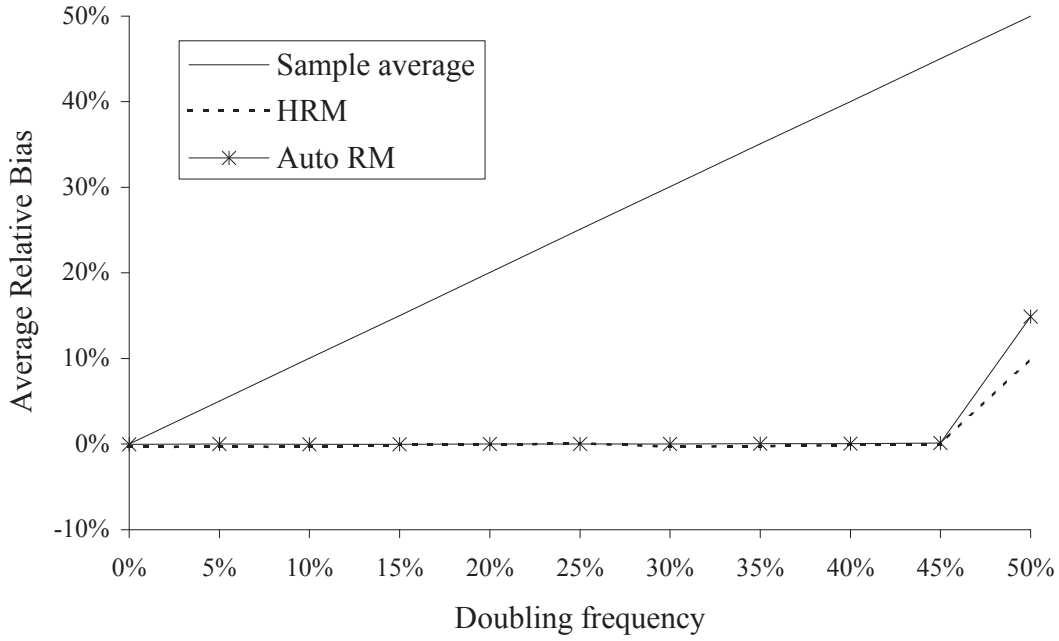


Figure 8.16: Bias comparison of item average estimators for symmetric data.

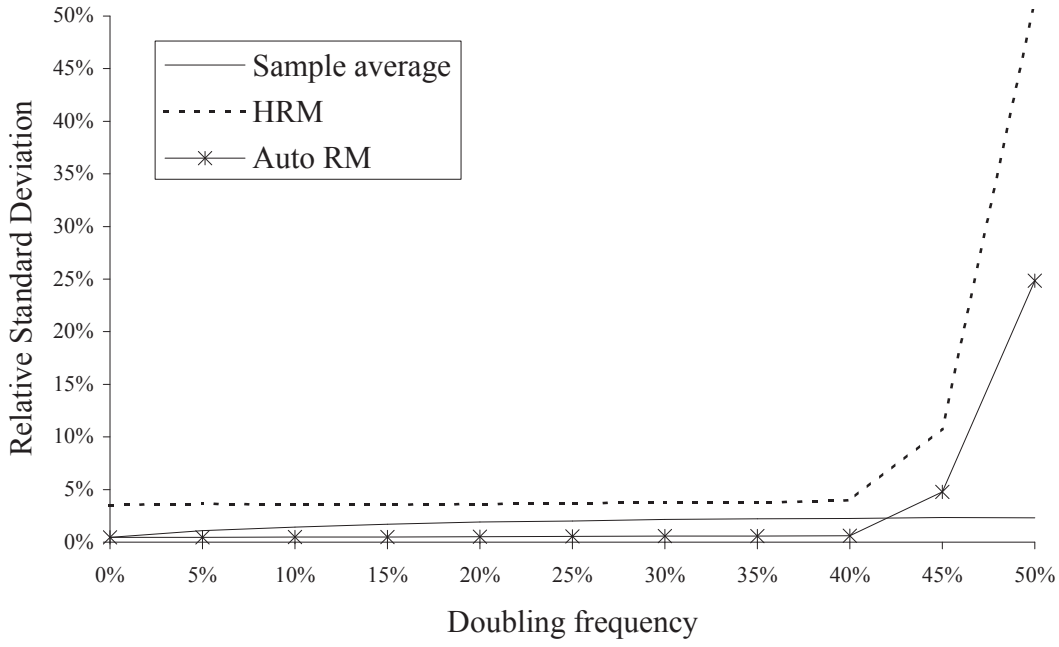


Figure 8.17: Standard deviation comparison of item average estimators for skewed data.

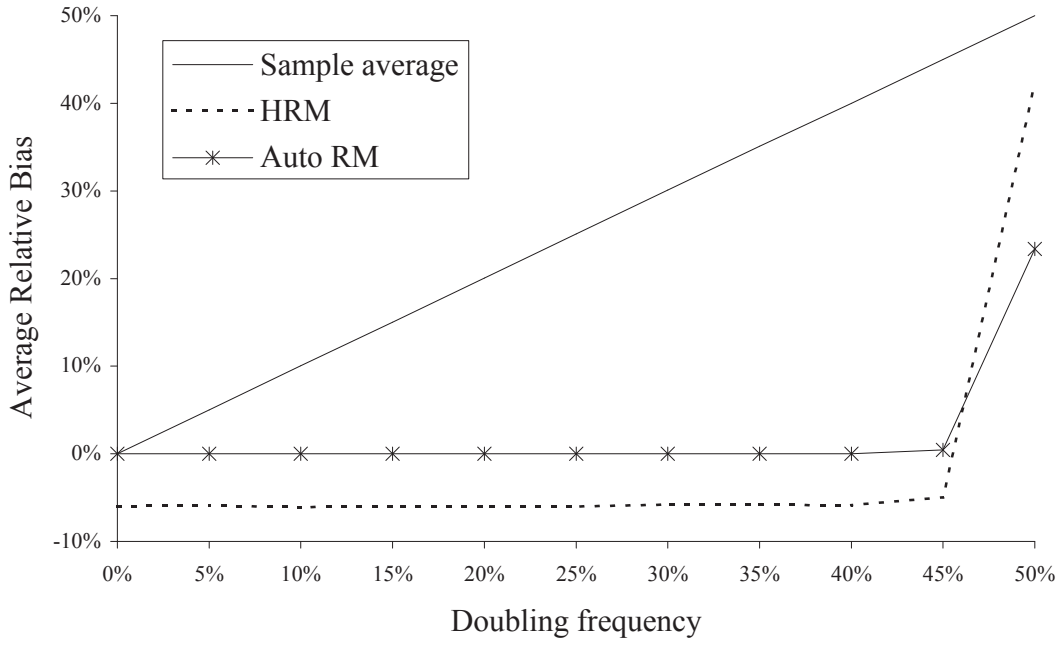


Figure 8.18: Bias comparison of item average estimators for skewed data.

February 9th, 2005, from 7:34 AM through 1:06 PM. During that time 29,963 pieces were logged, with an average weight of 122.07g, standard deviation 31.58g, minimum 51.7g, and maximum 446.8g. By visual inspection of the distribution in Figure 8.8, and practical knowledge of the product, it is assumed that items of 180g and above most likely all represent more than one item weighed together. This assumption is used for the first test, shown in Figure 8.19, where the doubling rate as estimated by the Auto RM Algorithm is plotted against the ratio of items that are 180g or more. Auto RM has $\rho = 70\%$, and the size of the FIFO queue is 500. The doubling estimate is calculated by subtracting 1 from the values given by the average number of items per slot ($\hat{\mu}/\tilde{\mu}$) using the Auto RM estimate ($\tilde{\mu}$) and sample average ($\hat{\mu}$). The data is run sequentially as logged and for each 500 items (one refill of the FIFO queue), the Auto RM estimate is calculated as well as the ratio of items above 180g. The Auto RM estimates track the solid line that represents perfect fit quite well, indicating that the algorithm is performing very well.

Figure 8.20 shows how the sample item average compares to the item average as estimated by Auto RM. Looking at the sample item average (dotted line) indicates that the product starts out quite heavy, and then drops in weight significantly during the monitored time. An examination of the Auto RM item average (solid line) leads to a different conclusion, specifically, that the item average is fairly stable during the entire time, but seems to drop just slightly over time from 117–122 to 110–120. The difference is the doubling, as plotted in Figure 8.21. Doubling is significantly higher during the first part of the series, as compared to the second half.

8.3 On-Line Item Distribution Estimation and Prospect Function Calculation

In practical applications, the item distribution is not known, and has to be estimated. This is easily done by keeping a FIFO queue of the weights of the last N items, and calculating a histogram from that. The item weights are real numbers, so an interval

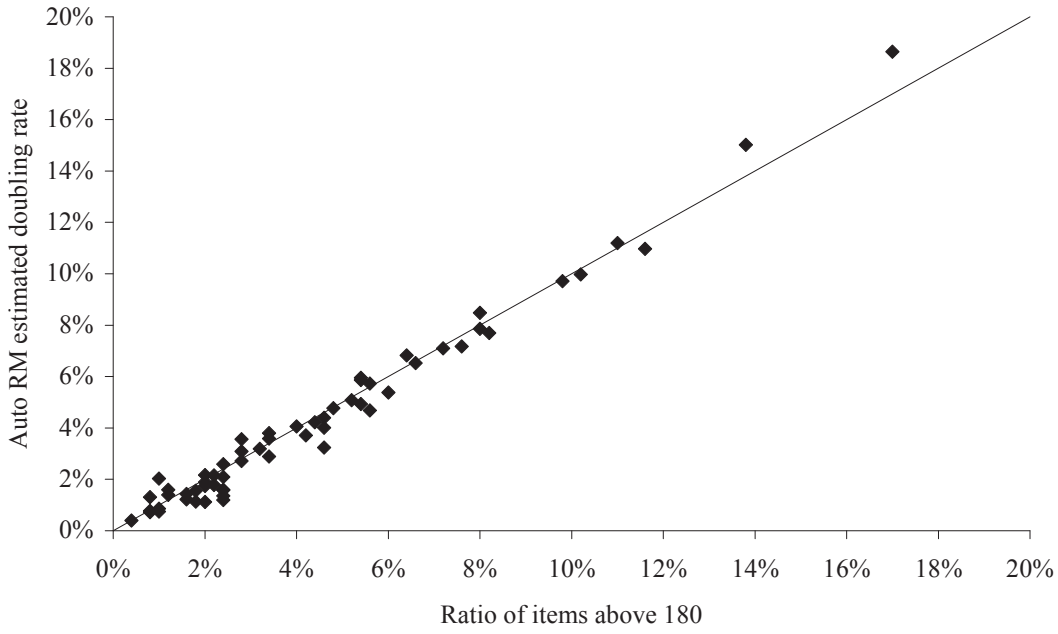


Figure 8.19: Real-world test I of Auto RM using Lufkin data. Doubling rate according to Auto RM vs. ratio of items that weigh above 180 (solid line represents perfect fit).

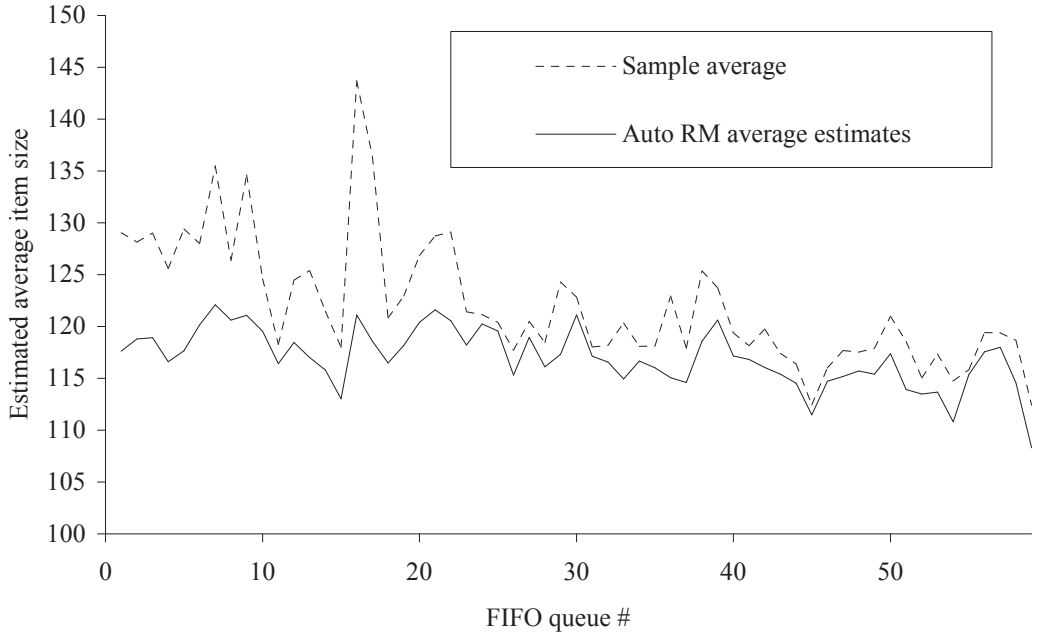


Figure 8.20: Real-world test II of Auto RM using Lufkin data. Average estimated using Auto RM compared to sample average.

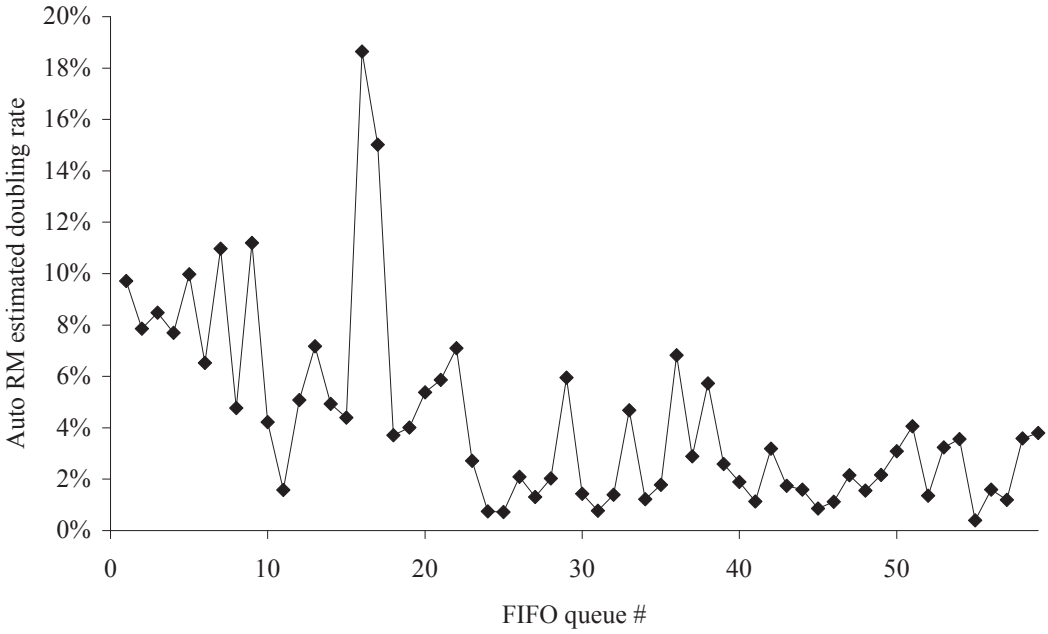


Figure 8.21: Real-world test III of Auto RM using Lufkin data. Doubling rate as measured by Auto RM, one estimate per 500 items.

size for the histogram has to be defined. The scale has a known weight accuracy, and it convenient to use that value as the interval size as well. Typically, the weighing accuracy is 1g, 2g, 5g, or 10g, depending on the physical properties and sizes of the items. The following table gives rough estimates of the weighing accuracy of Marel graders as a function of item size, where the accuracy is usually 2% of the smallest item a grader is designed to weigh:

Accuracy	Item size
1g	50g–500g
2g	100g–1000g
5g	250g–2,500g
10g	500g–5,000g
20g	1,000g–10,000g

As in the previous section, the length of the FIFO queue used to estimate the item size distribution, N, is a parameter that needs to be set in a balanced fashion. If N is too small, the histogram is not going to be very precise; and if N is too large, then the histogram will change too slowly if the input distribution changes. Estimating the distribution is not the only issue, however; it is also necessary to ensure that the derived Prospect function used for the Prospect Algorithm is kept up-to-date. This is more complicated because of the calculation effort needed. *Recursive Prospect* is defined as the Prospect function that results when the recursive formulation of Equation (3.7) is used to calculate it. Program 8.3.1, *Recursive Prospect Calculation*, shows a simple implementation using the R programming language [115]. The parameter B is the "length" of the Prospect function. To cover all possible lookups in the Prospect function, B has to be equal to:

$$B = b + f_{\max} - 1, \tag{8.4}$$

where b is the bin size and $f_{\max}$ is the maximum item size ($b+f_{\max}-1$ is the largest bin that one could possibly make). Note that since subscripts are not available in R-code, the minimum and maximum item sizes are represented as fn and fx, respectively.

```
RecursiveProspect <- function(B, fh, fn, fx)
{ #Inputs:
  #Length of prospect: B
  #FIFO histogram: fh
  #Minimum and maximum item sizes: fn, fx

  #Create a vector to store the prospect. Size is B+1 since R
  #has one as the minimum index. Need therefore to offset by
  #one when indexing. All variables are automatically
  #initialized as zero.
  P = mat.or.vec(B+1, 1)
  P[1] = 1
  #Calculate the prospect with recursive convolution.
  #Outer loop is over the length of the prospect.
  #Starts at fn, above 0 and below fn the prospect is zero.
  for (y1 in fn:B) {
    #Inner loop is bound by the distribution (within fn and fx).
    for (y2 in fn:min(y1,fx)) {
      P[y1+1] = P[y1+1] + fh[y2]*P[y1-y2+1]
    }
  }
  return(P)
}
```

Program 8.3.1: Recursive Prospect Calculation, an R-program for calculating the Prospect function according to Equation (3.7).

The number of loops needed to calculate the Prospect function with Recursive Prospect Calculation is given by:

$$(R_f + 1)\left(B + 1 - f_{\min} + \frac{R_f}{2}\right), \tag{8.5}$$

where $f_{\min}$ is the minimum item size and $R_f = f_{\max} - f_{\min}$ is the item range. Equation (8.5) is derived in Appendix F. This can represent a significant number of loops and, depending on the available processing power, it can take too much effort to calculate the entire Prospect function for each item added. One way to address this issue is to update only a part of the Prospect function at each item, but the cost of doing that is that the algorithm becomes slower to adapt to changes in the item distribution. This effect will be analyzed in the simulations in Section 8.4.

The Prospect function can also be calculated using Equations (3.4) and (3.5). *Count Prospect* is defined as the Prospect function that results when the recursive formulation of Equations (3.4) and (3.5) is used to calculate it. In that case, an extra count parameter, C, is needed to set the maximum number of summed items that are handled via convolution of the item distribution in Equation (3.4). Prospects of sums of more than C items are calculated with the Central Limit approximation of Equation (3.5). Program 8.3.2, *Count Prospect Calculation*, shows how the pmf $f^{(c)}(w)$ can be calculated for $c \in \{2, \ldots, C\}$ (here $f^{(c)}(w) = \mathrm{P[w,c]}$). The results of the program can be used, with the Central Limit Theorem (CLT) function estimate in Equation (3.5) for $c > C$, to calculate the prospect of a bin according to Equation (3.4). The program is written in the R programming language. The number of loops needed to compute all the pmf's for the Count Prospect calculation is given by:

$$(R_f + 1)(C - 1)\left(1 + R_f\frac{C}{2}\right), \tag{8.6}$$

where R_f is the item range, as in Equation (8.5). Equation (8.6) is derived in Appendix F.

The number of loops needed to calculate the pmf's for the Count Prospect calculations is similar in magnitude as that of the Recursive Prospect calculation (see Table 8.4 on Page 188), and therefore it also can be too much effort to calculate the entire Prospect function for each item added in practical applications. Summing up the pmf's to get the Count Prospect (see Equation (3.4)) is a quick operation that does not factor in this analysis.

One obvious way to reduce the number of loops needed to keep an up-to-date pmf is to skip calculating any convolutions, or set $C = 1$. The distribution of sums of $C > 1$ items is then estimated by the continuous approximation of Equation (3.5) (CLT). This is fine if the distribution being used is close to a bell curve, but if the distribution differs significantly from a Gaussian distribution, there might be a penalty for skipping the convolutions since, for example, the distribution of a sum of two items can be significantly different than from a bell curve. Figure 8.22 shows an example of this, where the distribution of the sum of two items from the twin-peaked distribution of Figure 8.2 is compared to the corresponding CLT approximation that is only based on the distribution's mean and variance. Setting an appropriate value for C is examined using simulation in Section 8.4.

Count Prospect pmf calculations can be made more efficient by *updating* the convolutions of Equation (3.4), instead of recalculating them, when an item gets added to the FIFO queue. This is called a *Tally Calculation*. The key is to keep the histogram and resulting convolutions in the Count Prospect as an *item combination count*, not as a probability. This makes it is possible, with relatively few loops, to keep a *Tally* of the number of combinations of possible sums of the items in the FIFO queue. The Tally can be updated for each item added and dropped from the FIFO queue; it is not necessary to calculate everything from scratch.

As an example, Table 8.2 shows the number of combinations of 1, 2, and 3 items that can yield different item sums for the item FIFO queue $\{2_1, 3_2, 4_3, 3_4, 2_5, 3_6\}$, where

```
CountProspect <- function(C, P, fn, fx)
{ #Inputs:
  #Maximum number of exact counts: C
  #FIFO histogram and result matrix: P
  #Minimum and maximum item sizes: fn, fx

  #Calculate the prospect with (C-1) convolutions.
  #Size of P is [C*fx,C], first column P[,1] should contain the
  #item histogram, the other columns should be initialized to
  #zero. The colums P[,2] to P[,C] are calculated by convolution
  #from P[,1]. The max/min bounds in the innermost loop ensure
  #that only potentially nonzero entries of P[,1] and P[,x] are
  #used as the weight y2 of one item is between fn and fx and
  #the weight y1-y2 of x items is between x*fn and x*fx.
  for (x in 1:(C-1)) {
    for (y1 in ((x+1)*fn):((x+1)*fx)) {
      for (y2 in max(fn,y1-x*fx):min(fx,y1-x*fn)) {
        P[y1,x+1] = P[y1,x+1] + P[y2,1]*P[y1-y2,x]
      }
    }
  }
  return(P)
}
```

Program 8.3.2: Count Prospect Calculation, an R-program for calculating the pmf $f^{(c)}(w)$ for Equation (3.4).

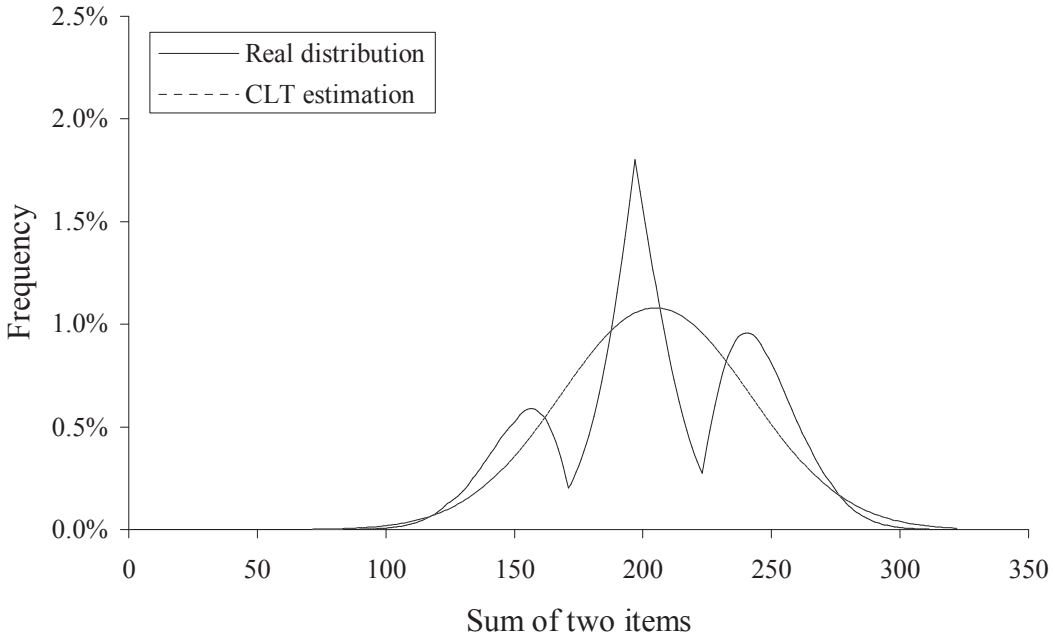


Figure 8.22: CLT approximation versus the real distribution of the sum of two items. Distribution is from Figure 8.2.

the value denotes the size and the subscript the position in the FIFO queue (the lower the subscript, the older the item is). This is called the *Tally Matrix*. The columns of the Tally Matrix represent the number of items, labeled N, and the rows represent the total weights of the items, labeled W. Columns are numbered 1 through C, where C is the maximum number of items summed together in the Tally Matrix; $C = 3$ in Table 8.2. Since the minimum item size possible is 1 ($f_{\min} = 1$), the rows run from 1 through the maximum number of items times the maximum item size $f_{\max}$, or $Cf_{\max}$. A location in the Tally Matrix is denoted by its W–N row/column location (w, n). Each item of the FIFO queue can be used more than once in each combination, or there can be more than one instance of each item in each combination Tally. As an example, the boldfaced number of 13 at location $(6, 2)$ in Table 8.2 represents the

Table 8.2: Tally Matrix for FIFO queue $\{2_1, 3_2, 4_3, 3_4, 2_5, 3_6\}$.

		Number of items N		
		1	2	3
Weights W	1			
	2	2		
	3	3		
	4	1	4	
	5		12	
	6		**13**	8
	7		6	36
	8		1	66
	9			63
	10			33
	11			9
	12			1
Total:		$6^1 = 6$	$6^2 = 36$	$6^3 = 216$

total number of combinations of two items that sum to 6, which are spelled out:

1: $\{2_1, 4_3\}$ 2: $\{4_3, 2_1\}$ 3: $\{2_5, 4_3\}$ 4: $\{4_3, 2_5\}$ 5: $\{3_2, 3_2\}$

6: $\{3_2, 3_4\}$ 7: $\{3_4, 3_2\}$ 8: $\{3_2, 3_6\}$ 9: $\{3_6, 3_2\}$ 10: $\{3_4, 3_4\}$

11: $\{3_4, 3_6\}$ 12: $\{3_6, 3_4\}$ 13: $\{3_6, 3_6\}$

The last row in Table 8.2 shows the total number of combinations in each column.

The Tally Matrix can be used to compute the values of the pmf $f^{(c)}(w)$ in the Prospect Algorithm since it will be identical to the values calculated by Program 8.3.2 if the Tally is divided by the total number of combinations in each column. To keep the Tally Matrix up-to-date, two algorithms are needed — one when an item is dropped (or removed) from the queue and one when an item is added. These algorithms are called *Tally Drop* and *Tally Add*, respectively.

To aid in their description, the Tally Matrix in Table 8.2 is used as an example, where the first item (2_1) is dropped, and then a new item of size 4 added. The algorithms are also listed as R-code in Program 8.3.3 (Tally Drop) and Program 8.3.4 (Tally Add). Tally Drop (or Add) performs the update in two separate steps that

subtract (or add) the combinations that the dropped (or added) item is a part of. In Tally Drop, Step 1 is a single loop to subtract all combinations that result from just the dropped item (w in the R-code, 2_1 in the example) by itself, which is accomplished by looping a variable n from 1 through C and at each step subtracting 1 from the Tally Matrix at location (nw, n). In the example this corresponds to the locations $(2, 1)$, $(4, 2)$, and $(6, 3)$, which are reduced by 1 as shown in *Step 1* of Figure 8.23. After this first step, the first column of the Tally Matrix (which represents the item histogram) is fully updated, but the other columns are only partially updated.

The second step uses three nested loops to subtract the additional combinations that the dropped item is a part of. The first loop iterates its variable, n, over the number of items (columns) from 1 through $C - 1$. During a loop, the current column n is up-to-date, and after each loop the next column will be up-to-date. The second loop iterates its variable m over the columns that need to be updated. The innermost loop iterates its variable y over the rows of column n of the Tally Matrix, which represent the possible total weights of n items, from $nf_{\min}$ through $nf_{\max}$. A bookkeeping calculation is performed at each step in the innermost loop, that subtracts the number of combinations that are contributed to the Tally Matrix (TM) entry $TM(y + w(m - n), m)$ involving $(m - n)$ repetitions of the dropped item w. The number of combinations that are subtracted is given by:

$$TM(y, n)\binom{m}{n}, \text{ where } m \geq n. \tag{8.7}$$

The first part of Equation (8.7), $TM(y, n)$, is simply the number of base combinations of n items (not including the dropped item of weight w) that sum to y, and the second part, $\binom{m}{n}$, counts how many ways there are to choose which n out of m items do not include the dropped item of weight w. The tables marked *Step 2* in Figure 8.23 show the results of the second set of nested loops of Tally Drop, broken down by the values of n.

```
TallyDrop <- function(TM, w, C, fn, fx)
{ #Inputs:
  #Size of item dropped: w
  #Maximum number of exact counts: C
  #Tally matrix: TM
  #Minimum and maximum item sizes: fn, fx
  #Binom(a,b), where a>=b,  returns the binomial coefficient

  for (n in 1:C) #Step 1.
  { #Subtract combinations that resulted from selecting
    #the dropped item w repeatedly.  This updates column 1
    #which is the item weight histogram.
    TM[n*w,n]=TM[n*w,n]-1
  }

  for (n in 1:(C-1)) #Step 2
  { #Loop n over the columns that have already been updated.
    for (m in (1+n):C)
    { #Loop m over the columns to be updated.  The first column
      #is already up-to-date efter Step 1.
      for (y in (fn*n):(fx*n))
      { #Loop y over the nonzero rows of column n. Values in the
        #m column are updated one-by-one by subtracting the
        #combinations involving (m-n) occurrences of the dropped
        #item w.
        #TM[y,n] counts how many different combinations of
        #a sum of n items (excluding the (m-n) occurrences of
        #the dropped item w) equal y.
        #Binom(m,n) counts how many ways there are to choose
        #which n out of m items do not include the dropped
        #item of weight w.
         TM[y+w*(m-n),m]=
            TM[y+w*(m-n),m]-TM[y,n]*Binom(m,n)
      }
    }
  }
  return(TM)
}
```

Program 8.3.3: Tally Drop, an R-program for dropping an old item from the Tally Matrix.

Step 1:

	1	2	3
1			
2	2-1=1		
3	3		
4	1	4-1=3	
5		12	
6		13	8-1=7
7		6	36
8		1	66
9			63
10			33
11			9
12			1

Step 2, $n = 1$:

	1	2	3
1			
2	1		
3	3		
4	1	3-1∗2=1	
5		12-3∗2=6	
6		13-1∗2=11	7-1∗3=4
7		6	36-3∗3=27
8		1	66-1∗3=63
9			63
10			33
11			9
12			1

Step 2, $n = 2$:

	1	2	3
1			
2	1		
3	3		
4	1	1	
5		6	
6		11	4-1∗3=1
7		6	27-6∗3=9
8		1	63-11∗3=30
9			63-6∗3=45
10			33-1∗3=30
11			9
12			1

Figure 8.23: Tally Matrix update when dropping item 2_1 from FIFO queue $\{2_1, 3_2, 4_3, 3_4, 2_5, 3_6\}$.

Tally Add is similar to Tally Drop; the difference lies in the order in which the steps and nested loops are performed. The first step uses three nested loops to incorporate the additional combinations that the added item (w in the R-code, 4 in the example) is a part of. The first loop iterates its variable, m, over the number of items (columns) backwards from C down to 2, specifying the column that is being updated. The second loop iterates its variable n over the columns from 1 to $m-1$, or the columns that have not been updated yet. The innermost loop iterates its variable y over the rows of column n of the matrix, which represent the sum of weights of n items, from $nf_{\min}$ through $nf_{\max}$. The same bookkeeping calculation is performed as in Tally Drop, except now the values from Equation (8.7) are added, not subtracted. The second steps adds the combinations that result from adding the new item w repeatedly. Figure 8.24 shows how the Tally Matrix is updated when adding a new item of size 4 to the demonstration FIFO queue (after the first item has been dropped as in Figure 8.23).

A limiting factor for the Tally Matrix calculation is the maximum size of the integer type that is used to store each Tally. The maximum size needed for an individual cell in the Tally Matrix is dependent on both the size of the FIFO queue N and the maximum count C, and is equal to N^C (corresponding to the case when all items of the FIFO queue are identical; e.g., say the only item size is b, so that the number of permutations that make the sum of C items equal $b \times C$ is N^C). The integer type used in the Tally Matrix needs to be able to store that number. Table 8.3 shows the resulting maximum FIFO queue size for 32-bit and 64-bit unsigned integers as a function of C. Modern computers use either 32-bit or 64-bit integers as a default minimum storage unit. Using the default unit of the used computer is most efficient. For example, if the FIFO size should be at least 255 entries, the maximum count C is going to be 4 and 8 for 32-bit and 64-bit integers, respectively. The number of loops needed to update the Tally Matrix (a Tally Drop followed by a Tally Add) is given by

```
TallyAdd <- function(TM, w, C, fn, fx)
{ #Inputs:
  #Size of item added: w
  #Maximum number of exact counts: C
  #Tally matrix: TM
  #Minimum and maximum item sizes: fn, fx
  #Binom(a,b), where a>=b,  returns the binomial coefficient

  for (m in C:2) #Step 1.
  { #Loop m over the columns to be updated backwards.  The first
    #column will not be be updated until Step 2.
    for (n in 1:(m-1))
    { #Loop n over the columns that have not been updated yet.
      for (y in (fn*n):(fx*n))
      { #Loop y over the nonzero rows of column n. Values in the
        #m column are updated one-by-one by adding the
        #new combinations involving (m-n) occurrences of the
        #added item w.
        #TM[y,n] counts how many different combinations of
        #a sum of n items (excluding the (m-n) occurrences of
        #the added item w) equal y.
        #Binom(m,n) counts how many ways there are to choose
        #which n out of m items do not include the added
        #item of weight w.
        TM[y+w*(m-n),m]=
          TM[y+w*(m-n),m]+TM[y,n]*Binom(m,n)
      }
    }
  }

  for (n in 1:C) #Step 2
  { #Calculate extra combinations that result from selecting
    #new item w repeatedly.  This updates column 1 which equals
    #the item weight histogram and completes the updates of the
    #other columns.
    TM[n*w,n]=TM[n*w,n]+1
  }
  return(TM)
}
```

Program 8.3.4: Tally Add, an R-program for adding a new item to the Tally Matrix.

Step 1, $m = 3$ and $n = 1$:

	1	2	3
1			
2	1		
3	3		
4	1	1	
5		6	
6		11	1
7		6	9
8		1	30
9			45
10			30+1∗3=33
11			9+3∗3=18
12			1+1∗3=4

Step 1, $m = 3$ and $n = 2$:

	1	2	3
1			
2	1		
3	3		
4	1	1	
5		6	
6		11	1
7		6	9
8		1	30+1∗3=33
9			45+6∗3=63
10			33+11∗3=66
11			18+6∗3=36
12			4+1∗3=7

Step 1, $m = 2$ and $n = 1$:

	1	2	3
1			
2	1		
3	3		
4	1	1	
5		6	
6		11+1∗2=13	1
7		6+3∗2=12	9
8		1+1∗2=3	33
9			63
10			66
11			36
12			7

Step 2:

	1	2	3
1			
2	1		
3	3		
4	1+1=2	1	
5		6	
6		13	1
7		12	9
8		3+1=4	33
9			63
10			66
11			36
12			7+1=8

Figure 8.24: Tally Matrix update when adding an item of size 4 to FIFO queue $\{3_1, 4_2, 3_3, 2_4, 3_5\}$.

Table 8.3: Maximum FIFO queue size N for Tally Matrix calculations when using 32-bit or 64-bit unsigned integers for storing the entries.

C	32-bit	64-bit
2	65535	4294967295
3	1624	2642244
4	255	65535
5	83	7130
6	39	1624
7	22	564
8	15	255
9	10	137
10	8	83

the following equation, assuming that both the minimum and maximum items ($f_{\min}$ and $f_{\max}$) in the FIFO queue stay constant:

$$\frac{C(C+1)}{3}[R_f(C-1)+3]. \tag{8.8}$$

Equation (8.8) is derived in Appendix F. Equation (8.8) results in fewer loops compared to Equations (8.5) and (8.6) for reasonable values of R_f, B, and C, as seen in the comparison Table 8.4. This shows that the Tally calculation is significantly more efficient than both the Recursive and Count Prospect calculations. As for the Count Prospect, summing up the pmf's calculated by the Tally method is a quick operation, and does not affect the results of this analysis. The question is whether the limit it imposes on the size of C is a problem, which will be investigated in the simulations in the next section.

8.4 Setting Parameters for On-Line Item Distribution Estimation and Prospect Function Calculation

Depending on the chosen type of Prospect function — Recursive or Count Prospect — there are one or two parameters that need to be set in addition to the zone parameter. Both need to set N, the length of the FIFO queue of recent items that is used as an estimate of the item distribution. For Count Prospect, the maximum number of exact

Table 8.4: Approximate number of loops needed to recalculate/update a Prospect structure using the Recursive Prospect calculation, Count Prospect calculation, or Tally method.

B	C	Distribution	$f_{\min}$	$f_{\max}$	R_f	Recursive Prospect	Count Prospect	Tally
400	3	N_D(100,15)	55	145	90	27391	24752	732
800	3	N_D(100,15)	55	145	90	63791	24752	732
1200	3	N_D(100,15)	55	145	90	100191	24752	732
400	5	N_D(100,15)	55	145	90	27391	82264	3630
800	5	N_D(100,15)	55	145	90	63791	82264	3630
1200	5	N_D(100,15)	55	145	90	100191	82264	3630
400	3	N_D(100,20)	39	161	122	37023	45264	988
800	3	N_D(100,20)	39	161	122	86223	45264	988
1200	3	N_D(100,20)	39	161	122	135423	45264	988
400	5	N_D(100,20)	39	161	122	37023	150552	4910
800	5	N_D(100,20)	39	161	122	86223	150552	4910
1200	5	N_D(100,20)	39	161	122	135423	150552	4910

convolutions, C, needs to be set as well. In all the simulations in this section, the improved Prospect+ Algorithm of Section 5.5 is used. First we examine the choice of C.

To examine the effects of C when using the Count Prospect function, we generated three sets of non-Gaussian distributions. One set is skewed to the left (long left tail or negative skewness), one set is skewed to the right (long right tail or positive skewness), and one set is twin-peaked (items in the middle missing). All the distributions are generated by filtering out a range of items from a bigger (discrete) Gaussian distribution and then adjusting the resulting distribution, so that the distributions all have approximately the same number of items (values), about 2000 (see Appendix A), the same average, and the same standard deviation as the corresponding Gaussian distribution. Their range (min and max) is, however, different from the range of the corresponding Gaussian distribution. The method for generating the distributions was ad-hoc, and the details are not important for this discussion, but

Table 8.5: Comparison of non-Gaussian and Gaussian distribution statistics.

Name	μ	σ	Min	Max	Skewness	Kurtosis
N_D(100,10)	100.0	10.01	68	132	0	-0.055
LS(100,10)	100.0	10.00	63	115	-0.645	-0.005
RS(100,10)	100.0	10.00	85	137	0.645	-0.005
TP(100,10)	100.0	10.00	74	126	0	-1.203
N_D(100,15)	100.0	14.86	55	145	0	-0.126
LS(100,15)	100.0	14.85	44	120	-0.853	0.342
RS(100,15)	100.0	14.85	80	156	0.853	0.342
TP(100,15)	100.0	14.86	70	130	0	-1.733
N_D(100,20)	100.0	20.01	39	161	0	-0.064
LS(100,20)	100.0	20.00	33	132	-0.550	-0.187
RS(100,20)	100.0	20.00	68	167	0.550	-0.187
TP(100,20)	100.0	20.00	61	139	0	-1.764

the distributions are listed in detail in Appendix A. The distributions skewed to the left are labeled LS(μ,σ), the ones skewed to the right are labeled RS(μ,σ), and the twin-peaked ones are labeled TP(μ,σ), where μ and σ are the mean and standard deviation of the item distribution, respectively. The item average is set at 100, and the standard deviations at 10, 15, and 20, so that they can be better compared to their discrete Gaussian counterparts that have been used frequently in this thesis. Table 8.5 summarizes the statistics of the distributions with their discrete Gaussian counterparts.

To generate the comparisons, we performed a series of simulations using the distributions of Table 8.5. Each simulation starts with a warmup period until 100 bins have been filled, and after that the simulation is run until $1{,}000 \times 30 = 30{,}000$ bins have been generated. The bins are grouped into 30 batches of 1,000 bins, and the average overfill is calculated for each one. This is then used to calculate the average overfill and standard deviation, s, of the overfill. There are eight active bins, the bin size is varied from 200 to 800 in increments of 10, and the optimal zone size for each simulation is determined with the Optimal Zone Search Algorithm 3.3. The length of

the FIFO queue, N, is set to 500. For each distribution, the average overfill over all the different bin sizes is calculated, and its 95% confidence interval calculated under the assumption that the calculated overfill for each bin size is normally distributed with the square of the calculated standard deviation, s^2, used as an estimate for the variance. Each simulation is run seven times for different Prospect function calculations, i.e., once using the Recursive Prospect, and six times using the Count Prospect with C varied from one through six in increments of one. Tables 8.6 through 8.8 and Figures 8.25 through 8.27 summarize the results.

It is expected that the type of Prospect function will not have an effect if the item distribution is approximately Gaussian, and that can be observed from the figures and tables. There is no significant difference in overfill observed for any of the discrete Gaussian distributions.

It is also expected that the Recursive Prospect function usually has the best performance since no approximations are used, and that the effect of using the Central Limit Theorem Equation (3.5) should be increasing with a decreasing value of C. This is however not the case for the narrowest (low relative variance) set of distributions in Table 8.6 and Figure 8.25. This can be explained by the low relative variance of the item distribution. For item distributions with low variance, the (bin size)/(item average) ratio is often the most-significant factor for determining the overfill, especially if there are few items needed to fill the bin on average. In those cases, the different Prospect function calculations do not significantly influence the results, so various other random effects determine the small difference that there is. If there is a significant difference in the performance, the Count Prospect version has the upper hand four times, but the Recursive Prospect only three times.

For the wider two sets of distributions, the expected pattern is visible for the left-skewed and twin-peak distributions, but not strongly observed for the right-skewed

Table 8.6: 95% confidence intervals of average overfill as a function of Prospect function type and item distribution with $\mu = 100$ and $\sigma = 10$. Boldfaced ranges indicate that the average overfill using Count Prospect is significantly higher than using Recursive Prospect and italic ranges indicate that the average overfill is significantly lower.

Prospect function C	LS(100,10)	RS(100,10)	TP(100,10)	N(100,10)
Count Prospect 1	**[18.52,18.60]**	[14.55,14.59]	*[16.11,16.16]*	[16.18,16.24]
Count Prospect 2	[18.21,18.29]	[14.53,14.57]	*[16.14,16.19]*	[16.18,16.23]
Count Prospect 3	**[18.38,18.46]**	*[14.47,14.51]*	*[16.27,16.32]*	[16.18,16.24]
Count Prospect 4	**[18.41,18.50]**	[14.54,14.58]	[16.36,16.40]	[16.17,16.23]
Count Prospect 5	[18.35,18.43]	[14.58,14.63]	[16.38,16.43]	[16.16,16.22]
Count Prospect 6	[18.29,18.38]	[14.58,14.62]	[16.35,16.40]	[16.17,16.23]
Recursive Prospect	[18.27,18.35]	[14.54,14.58]	[16.35,16.40]	[16.17,16.23]

distributions (see Figures 8.26 and 8.27). This indicates that the Central Limit Approximation of Equation (3.5) has a smaller effect if the left tail of the distribution (corresponding to small items) is missing. For the left-skew distributions, setting C at 4 or even 3 seems to be fine, and for the twin-peaked distributions, setting $C = 6$ eliminates the Central Limit Approximation effect. This indicates that the size limit on C imposed by the Tally Calculation method is not a problem, at least if the computer used has 64-bit integers available for use. If the computer used only has 32-bit integers available and therefore C has to be 3 or less (using Table 8.3 with $N = 500$), the penalty might not be great, and perhaps only significant if twin-peaked distributions are encountered. In our experience, left- or right-skewed distributions are much more common in practical situations than twin-peaked, so the impact of setting C below 6 may be very limited.

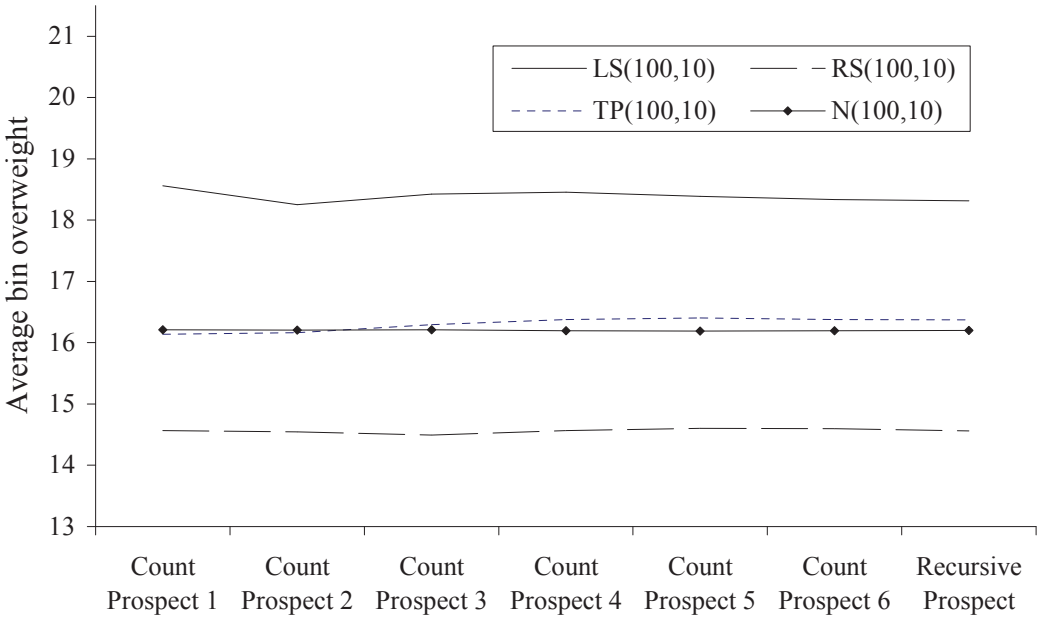


Figure 8.25: Average simulated overfill for each Prospect function type and item distribution with $\mu = 100$ and $\sigma = 10$.

Table 8.7: 95% confidence intervals of average overfill as a function of Prospect function type and item distribution with $\mu = 100$ and $\sigma = 15$. Boldfaced ranges indicate that the average overfill using Count Prospect is significantly higher than using Recursive Prospect and italic ranges indicate that the average overfill is significantly lower.

Prospect function C	LS(100,15)	RS(100,15)	TP(100,15)	N(100,15)
Count Prospect 1	**[10.69,10.76]**	[7.32,7.35]	**[12.03,12.06]**	[7.94,7.97]
Count Prospect 2	**[10.40,10.46]**	[7.35,7.37]	**[10.97,11.00]**	[7.94,7.97]
Count Prospect 3	**[10.34,10.39]**	**[7.37,7.39]**	**[10.56,10.59]**	[7.93,7.96]
Count Prospect 4	[10.27,10.33]	**[7.35,7.37]**	**[10.36,10.40]**	[7.93,7.97]
Count Prospect 5	[10.27,10.32]	[7.33,7.35]	**[10.26,10.29]**	[7.94,7.97]
Count Prospect 6	[10.25,10.31]	[7.32,7.34]	[10.22,10.25]	[7.94,7.97]
Recursive Prospect	[10.26,10.31]	[7.33,7.35]	[10.19,10.22]	[7.93,7.97]

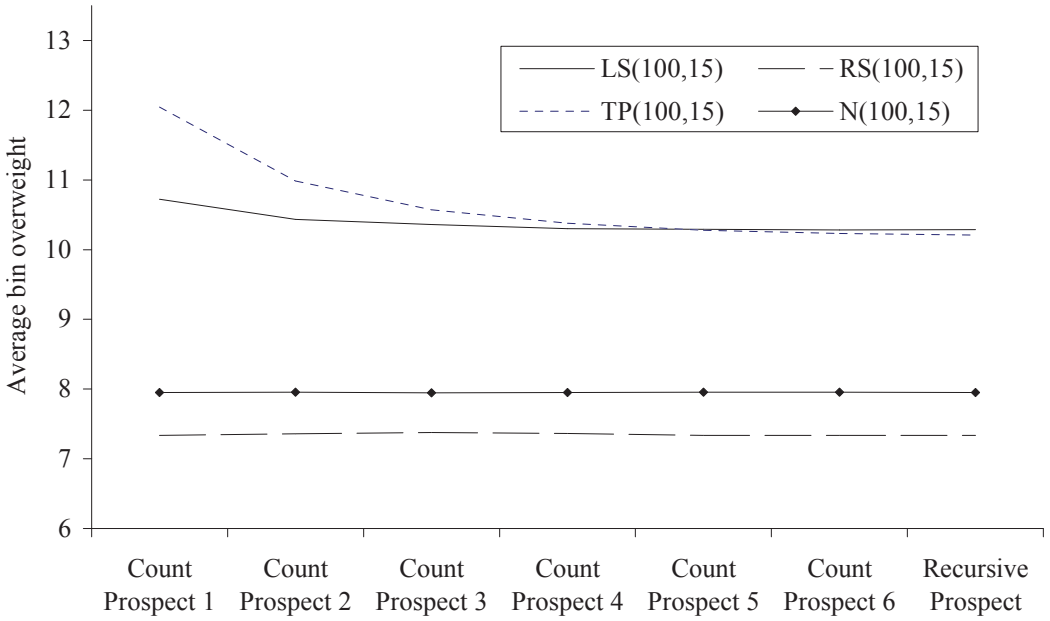


Figure 8.26: Average simulated overfill for each Prospect function type and item distribution with $\mu = 100$ and $\sigma = 15$.

Table 8.8: 95% confidence intervals of average overfill as a function of Prospect function type and item distribution with $\mu = 100$ and $\sigma = 20$. Boldfaced ranges indicate that the average overfill using Count Prospect is significantly higher than using Recursive Prospect and italic ranges indicate that the average overfill is significantly lower.

Prospect function C	LS(100,20)	RS(100,20)	TP(100,20)	N(100,20)
Count Prospect 1	**[6.48,6.54]**	*[4.50,4.52]*	**[11.08,11.11]**	[5.31,5.33]
Count Prospect 2	**[6.23,6.29]**	**[4.57,4.58]**	**[9.48,9.51]**	[5.30,5.33]
Count Prospect 3	[5.69,5.72]	**[4.57,4.58]**	**[8.72,8.75]**	[5.31,5.33]
Count Prospect 4	[5.71,5.75]	[4.56,4.57]	**[8.45,8.47]**	[5.31,5.34]
Count Prospect 5	[5.70,5.74]	[4.55,4.56]	**[8.33,8.35]**	[5.31,5.34]
Count Prospect 6	[5.70,5.74]	[4.54,4.56]	[8.27,8.30]	[5.31,5.33]
Recursive Prospect	[5.70,5.74]	[4.55,4.56]	[8.25,8.27]	[5.31,5.34]

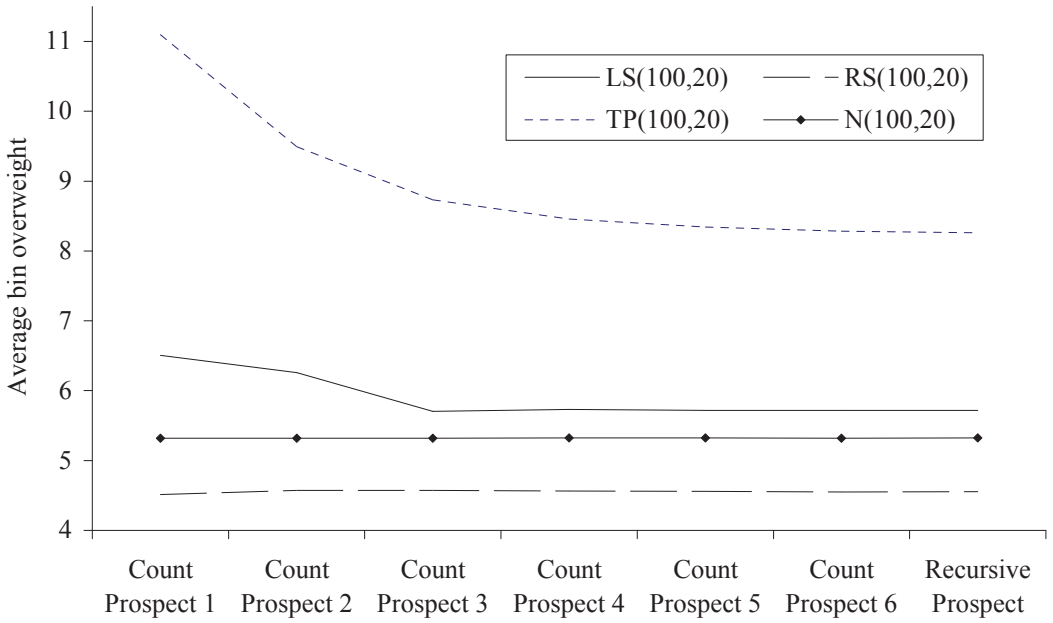


Figure 8.27: Average simulated overfill for each Prospect function type and item distribution with $\mu = 100$ and $\sigma = 20$.

Now we examine setting an appropriate value of N, the length of the FIFO queue of recent pieces that are used to estimate the item distribution, with simulations. The results are shown in Figure 8.28. There are two item distributions used in the simulations. Item distribution $\mathrm{N}_D(100,15)$ is used as a base and then a constant of 5 is either added to, or subtracted from, each item size, resulting in two item distributions that are either labeled $\mathrm{N}_D(95,15)$ or $\mathrm{N}_D(105,15)$. First, with pre-simulation, a bin size of 650 is selected because it has the same optimal zone size (2) for both input distributions, and very similar average overfill (≈ 1.05 and ≈ 1.15, respectively). The Count Prospect function is used with $C = 3$. There are five different lines in the figure representing the following different sets of simulations.

1 The solid flat line labeled *Known distribution* represents the case where the distribution is known exactly and the exact Prospect function can therefore be calculated at the start of the simulation. What is shown is the average of two simulations, one for each item distribution. Each of the two simulations is run until 50,000 bins have been filled, and the average overfill of the two simulations is plotted, as a function of N.

2 The slotted line with no markers labeled *No switching* shows the effect of estimating the distribution on the fly with a FIFO histogram of varying lengths. The input distribution is not oscillating, and what is shown is the average overfill of two simulations, one for each item distribution. Each of the two simulations is run until 50,000 bins have been filled.

3–5 The three lines with diamond markers show what happens when the item distribution oscillates, or switches, between the two input distributions. The interval between the switch varies, as either 5,000, 10,000, or 20,000 items. Graders typically process 125–250 pieces per minute, so this corresponds to 20–40 minutes, 40–80 minutes, and 80–160 minutes of production time, respectively. Each

simulation is run until 100,000 bins have been filled.

Figure 8.28 shows that the "no-switching" slotted line starts significantly higher than the solid line, but trends towards it as the size of the FIFO queue increases, since the accuracy of the distribution estimate increases. In the three cases with oscillating item distributions, the overfill slopes down towards the solid line similarly to the non–switching case when the FIFO queue length is small. As the FIFO queue size increases, the overfill drop slows and then reverses when the FIFO queue size gets to 1,000 or more. Thus the cost of increasing the FIFO queue length is that the memory of the Prospect function increases, and it becomes slower to adapt to changes in the input distribution. The optimal value varies with the switching frequency, but is in the range of 250 to 1,000. Figure 8.28 also shows that more-frequent changes in the item distribution lead to larger average overfill, as expected.

Finally, Figure 8.29 demonstrates the gain achieved by using the efficient Tally Calculations to keep the Prospect function up-to-date. The simulation setup is the same as for the 5,000 item distribution switching case from Figure 8.28. The difference is that for the slotted line the prospect is always kept up-to-date with the efficient Tally Calculation, but the solid line is calculated with the Recursive Prospect method similar to Program 8.3.1. Care is taken to keep the total calculation effort similar in the two approaches, to keep the comparison fair. With this in mind, the simulation is set so that 1.5% of the Recursive Prospect function is updated at each new item, and by doing that the Recursive Prospect simulation takes similar time as the Tally simulations. For example, if the length of the Prospect function vector is 1000, only 15 of those entries are calculated for each item. This is done sequentially, so for the following item the next 15 entries are calculated, and so on until the end of the array is reached and the calculation starts again at the beginning of the Prospect array. The difference in performance is clearly visible. Even though the lowest point of the solid curve is close to the slotted line, the solid curve is much more sensitive to changes in

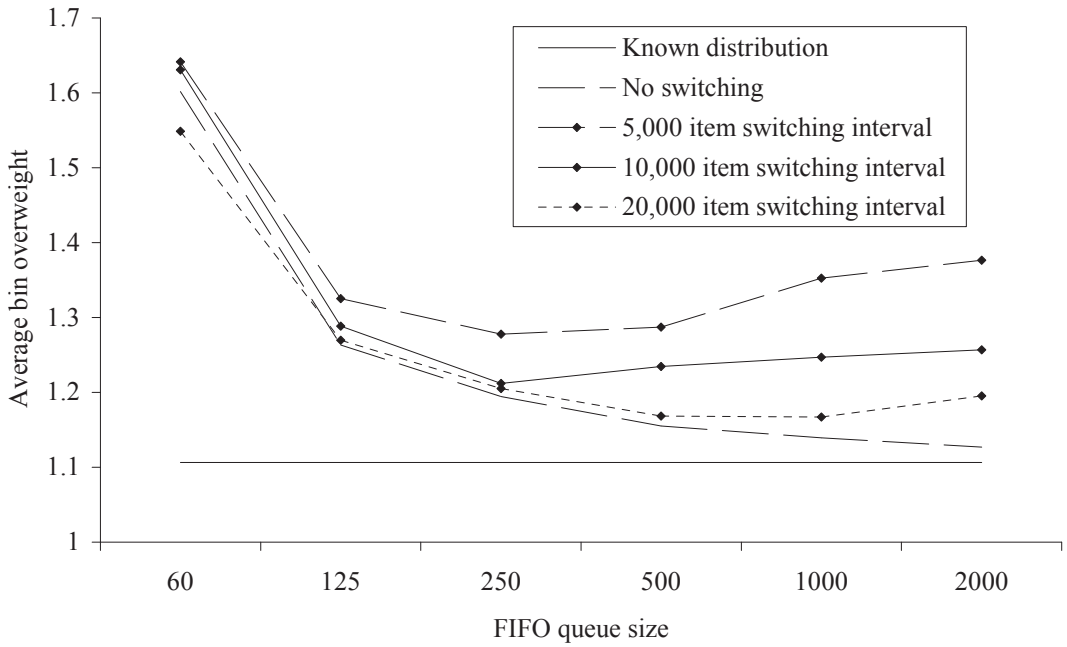


Figure 8.28: Effect of different FIFO queue sizes when using oscillating item distributions.

the FIFO queue size, making it harder to set a robust value.

8.5 Comparison of Different Versions of the Prospect Algorithm

In this section, different versions of the Prospect Algorithm set forth in this thesis are compared from a practical point of view. In practice, when creating packs of food products of fixed minimum weight, the average overfill is usually low. The producer gets no profit from the overfill since it is basically given away; hence there is strong incentive to make only products with low average overfill. If the item size/bin size combination does not allow low average overfill, those combinations will almost certainly not be made.

To simulate this effect, a series of pre-selection simulations were made where both the bin size and number of bins are varied. The original PR Algorithm 3.2 from Section 3.3 is used, and the optimal zone size for each simulation is determined with the Optimal Zone Search Algorithm 3.3. For each bin size, the simulation with the

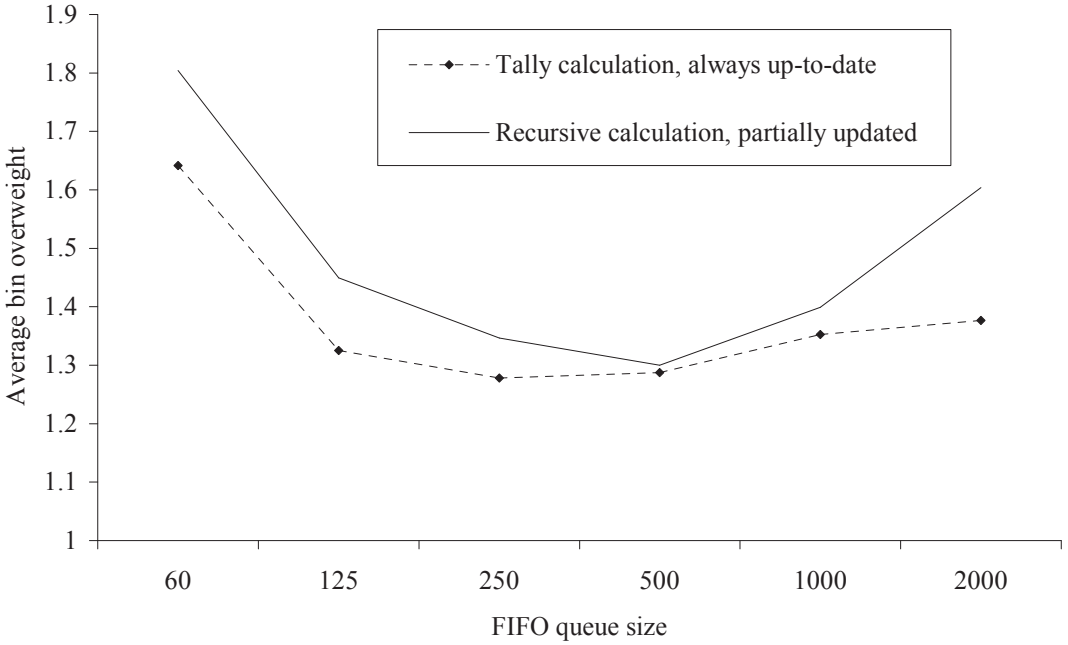


Figure 8.29: Example of the effect of using the efficient Tally Calculation to always keep the prospect up-to-date.

fewest bins that yielded an overfill below 2% of the item size was selected. The bin size is varied from 400 to 1200 in increments of 20, and the number of bins from 4 to 12 (even numbers only). The item distribution used is $N_D(100,15)$, and each simulation is run for 21,000 filled bins, where the first 1000 bins are discarded as warmup. Table 8.9 shows the results. Note that bin sizes 440, 460, and 480 are not included because the average overfill is above 2% for 12 bins, and hence these item size/bin size combinations do not allow for low-enough overfill to be practical. Also note that the number of bins reduces with bin size. This is reasonable because as bin size grows, it gets easier to fill bins with low overfill.

With the combinations of bin sizes and bin counts, a few different simulation comparisons are made comparing the two major alternatives to the original PR algorithm, namely, the PR+ algorithm of Section 5.5 and the PRE algorithm of Section 6.5. Those two algorithms are chosen since they represent the most-practical alternatives to the PR algorithm that are put forth in this thesis; the other algorithms

Table 8.9: Results of pre-selection simulations, PR algorithm, bin size 400 to 1200 in steps of 20, 4 to 12 active bins in steps of 2, $N_D(100,15)$ item distribution.

Bin size	Bins	Zone size	Over-fill	Items /bin
400	12	3	1.98	4.0
420	12	3	1.66	4.2
500	10	3	1.72	5.0
520	8	3	1.70	5.2
540	8	3	1.63	5.4
560	8	3	1.61	5.6
580	8	3	1.72	5.8
600	8	2	1.66	6.0
620	8	2	1.35	6.2
640	8	2	1.29	6.4
660	8	2	1.17	6.6
680	8	2	1.22	6.8
700	8	2	1.17	7.0
720	6	3	1.90	7.2
740	6	3	1.94	7.4
760	6	3	1.80	7.6
780	6	3	1.75	7.8
800	6	3	1.76	8.0
820	6	3	1.71	8.2

Bin size	Bins	Zone size	Over-fill	Items /bin
840	6	3	1.69	8.4
860	6	2	1.56	8.6
880	6	2	1.51	8.8
900	6	2	1.49	9.0
920	6	2	1.33	9.2
940	6	2	1.32	9.4
960	6	2	1.35	9.6
980	6	2	1.35	9.8
1000	6	2	1.23	10.0
1020	6	2	1.21	10.2
1040	6	2	1.23	10.4
1060	6	2	1.21	10.6
1080	6	2	1.14	10.8
1100	6	2	1.12	11.0
1120	6	2	1.10	11.2
1140	6	2	1.07	11.4
1160	6	2	1.11	11.6
1180	6	2	1.08	11.8
1200	6	2	1.05	12.0

either need more known items to work well (Section 7.3) or need more computations and are hence much slower (Section 6.7). First the average overfill is compared for the base case as in Table 8.9. The PR+ algorithm is compared with the same zone settings as the PR algorithm (the optimal zone is estimated with the Optimal Zone Search Algorithm 3.3), but for the PRE the discount parameter r is varied from 5% to 95% in steps of 5% and the optimal value selected for each bin size. The results are summarized in Table 8.10 and Figure 8.30.

It can be seen from these simulations that the PR+ always beats PR and PRE beats both in all cases. It should be noted, however, that PRE needs more calculations than PR/PR+, and is hence significantly slower. In the simulations for Tables 8.9 and 8.10, the speed difference is by a factor of about 2.5. The "jump" seen in all the overfill lines when the bin size hits 720 in Figure 8.30 is caused by the reduced

number of bins in the simulations when the bin size gets above 700. It goes from 8 down to 6, causing the jump. Since PR+ dominates PR, only PR+ and PRE are used from now on. In a real-world application one would simply use PR+ and not bother testing PR. The reason is that the PR and PR+ are closely related, in that the difference in their logic is not significant, and hence they behave similarly, except that PR+ is almost always a little better.

Table 8.10: Simulations as in Table 8.9 repeated for the PR+ and PRE versions of the prospect algorithm.

Bin size	Bins	PR+ Overf.	PRE r	PRE Overf.
400	12	1.77	0.45	1.73
420	12	1.51	0.35	1.43
500	10	1.63	0.35	1.36
520	8	1.58	0.35	1.42
540	8	1.54	0.35	1.32
560	8	1.54	0.30	1.18
580	8	1.59	0.35	1.31
600	8	1.33	0.30	1.14
620	8	1.15	0.30	1.04
640	8	1.09	0.30	1.00
660	8	1.06	0.25	0.98
680	8	1.05	0.20	0.93
700	8	1.04	0.20	0.94
720	6	1.72	0.35	1.54
740	6	1.71	0.35	1.48
760	6	1.67	0.30	1.47
780	6	1.63	0.35	1.39
800	6	1.61	0.30	1.31
820	6	1.59	0.30	1.32

Bin size	Bins	PR+ Overf.	PRE r	PRE Overf.
840	6	1.55	0.30	1.28
860	6	1.33	0.30	1.24
880	6	1.22	0.30	1.19
900	6	1.19	0.25	1.15
920	6	1.19	0.25	1.12
940	6	1.20	0.30	1.09
960	6	1.19	0.20	1.06
980	6	1.14	0.20	1.06
1000	6	1.12	0.25	1.04
1020	6	1.05	0.30	1.02
1040	6	1.08	0.20	1.02
1060	6	1.08	0.15	0.94
1080	6	1.05	0.15	0.94
1100	6	1.04	0.15	0.91
1120	6	1.05	0.15	0.77
1140	6	1.03	0.15	0.80
1160	6	1.03	0.15	0.76
1180	6	1.03	0.10	0.79
1200	6	1.01	0.15	0.71

In the rest of this section, three different practical scenarios are put forth and the algorithms tested by comparing to the original scenario ("base case") as shown in Table 8.10. All the scenarios test how sensitive the algorithms are to being run with suboptimal parameter settings. In all scenarios, the bin sizes used are the same as in Table 8.10. The first scenario is to test how sensitive the algorithms are to suboptimal settings of their parameters, namely, the zone z for PR+ and the discount factor r

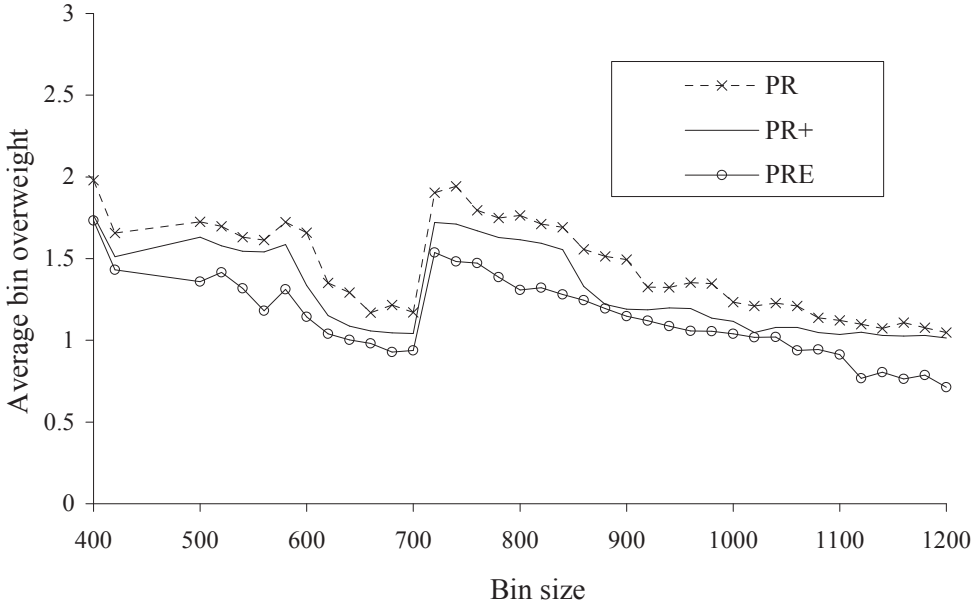


Figure 8.30: Comparison of overfill for the PR, PR+, and PRE algorithms.

for PRE. This is done by stipulating that a single value of the parameter be used in all simulations. The item data is again $N_D(100,15)$. For the PR+ algorithm, z varies from 1 to 3 in the simulation, and the best single value for all simulations is 2. For the PRE algorithm, the discount factor r varies from 0.10 to 0.45 in increments of 0.05, with the optimal single value being 0.30. This parameter choice is just for the first scenario; in the other scenarios the original parameter settings from Tables 8.9 and 8.10 are used. The second scenario tests how sensitive the algorithm is to changes in the item distribution; the item distribution is a modified $N_D(100,15)$ distribution; the average item size is moved by -4%, -2%, $+2\%$, and $+4\%$. The third and final scenario tests how sensitive the algorithms are to piece doubling as described in Section 8.1, with 2%, 4%, 6%, and 8% of pieces being doubled (again the underlying item distribution is $N_D(100,15)$). In the second and third scenarios, the parameters of the algorithms are potentially suboptimal since they were not optimized for the realized item distributions. The average overfill of all the simulations over all bin sizes under consideration is summarized in Table 8.11.

Table 8.11: Effect of suboptimal settings on the average overfill of the PR+ and PRE algorithms. The difference is positive if PRE has lower overfill than PR+.

	PR+	PRE	Difference
Base case (Table 8.10)	1.25	1.14	10.3%
Fixed parameter	1.29	1.24	3.6%
4% lighter items	1.25	1.08	16.2%
2% lighter items	1.27	1.11	14.6%
2% heavier items	1.48	1.56	-4.7%
4% heavier items	1.80	2.11	-14.4%
2% doubling	1.39	1.36	2.3%
4% doubling	1.51	1.56	-3.1%
6% doubling	1.95	1.95	-0.2%
8% doubling	2.21	2.46	-10.2%

PRE beats PR+ by 10.3% in the base case with optimal parameter settings for both, and again in the first scenario with the same parameter setting for all simulations, but by a much smaller margin of 3.6%. In the second scenario, PRE keeps its advantage if the items are lighter. Making the items lighter lowers the overfill because this increases the number of items per bin, which generally has a positive effect on the overfill since then there are more combinations of items that can be used to fill the bins. If the items are made heavier, the situation reverses, and PR+ has better performance. PR+ has also better overall performance with piece doubling, which is not surprising since piece doubling also makes the items that the algorithms "see" heavier on average. Thus the PRE advantage disappears quickly and reverses with increased doubling.

The conclusion is that while PRE has the potential to beat PR+, it can be more sensitive to suboptimal settings, making its advantage smaller. If one were to program a Prospect algorithm to work on a broad range of real-world problems, the appropriate strategy is to have both PR+ and PRE available, so the advantages of both can be utilized. The fact that PRE is slower to run is not that important in the real-world, because modern computer controllers can handle both PR+ and PRE with ease. Use

PRE if the environment is relatively stable, but perhaps switch to PR+ if the item distribution is likely to vary a great deal during production.

CHAPTER 9

SUMMARY, CONCLUSIONS, AND FURTHER RESEARCH

This chapter presents a summary of, and conclusions obtained from, the thesis, along with some recommendations for further research.

9.1 Summary and Conclusions

In this thesis a new algorithm for on-line Bin-Covering, the Prospect Algorithm, is introduced. The algorithm utilizes information available from the item distribution to minimize the average overfill. The algorithm can also be adapted to solve on-line Bin-Packing, as well as the following variations of the original Bin-Covering problem:

1. Some items can be rejected to improve the average overfill;
2. constraints on the number of items allowed in each batch;
3. multiple item types, where the goal is to have constraints on the number of items of each type in the batches;
4. multiple bin capacities, with the goal of finishing the bin close to one of several capacities.

Furthermore the Prospect Algorithm leads to some interesting calculations for other known problems:

1. With the probabilistic model used in the Prospect Algorithm, it is possible to calculate exactly the expected performance of the well-known Next-Fit (NF) algorithm [76] for Bin-Covering and Bin-Packing.

2. *Inspection paradox, renewal theory*: When observing a renewal process interval, the observed renewal interval distribution is typically larger than the average interval time. The same phenomenon applies to the distribution of the first item in a bin when using the NF algorithm for Bin-Packing. Using the Prospect function for Bin-Packing, it is possible to calculate the distribution of the first item in a bin, and hence the distribution of observed renewal times, for a discrete item (or renewal interval) distribution.

The Prospect Algorithm can also be adapted to work with *perfect packing*, which is a variation of regular Bin-Packing and Bin-Covering where there is not a constraint on the number of active bins, and bins are only finished by filling them exactly. The goal is to determine if the number of active bins is bounded when the number of items processed goes to infinity. We have shown with simulations that the Prospect Algorithm performs better than the well-known Sum-of-Squares algorithm. This analysis also leads to a slightly improved version of the Prospect Algorithm.

We also study the solution of on-line Bin-Covering problems with Markov decision processes (MDPs). First an optimal Bin-Covering strategy is calculated with an MDP (which is possible if the problem size is kept small). The overfill distribution of the optimal MDP strategy is analyzed and compared to the overfill distribution of the Prospect Algorithm. This leads to a modified Prospect Algorithm that performs better in some instances.

On-line Bin-Covering with lookahead is also studied. Lookahead means that the algorithm is on-line, but it can either store items for some time, or it has information about a few items ahead.

Finally, issues stemming from practical applications of the Prospect Algorithm are discussed. The assumption that the item distribution is known is dropped, so that it has to be estimated on-line, and the Prospect function recalculated constantly. The assumption that the distribution is either uniform or a discrete normal distribution is

relaxed as well. An improved and efficient calculation method to keep an up-to-date Prospect when the item distribution is changing is developed. The different versions of the Prospect Algorithms described in the thesis are compared in realistic settings.

9.2 Further Research

Below are a few ideas for future research:

1. It would be interesting to see how well the Prospect Algorithm would do if it were modified to solve Bin-Packing and Bin-Covering problems that are not on-line.

2. In the Perfect Packing chapter (Chapter 5), the Prospect Algorithm is compared to the *Sum-Of-Squares (SS)* algorithm. A striking optimality property of the SS algorithm has been proved, namely, that it will always achieve the optimal level of wasted space, and it would be interesting if a similar optimality property can be proved for the Prospect Algorithm, since simulations strongly indicate it does have this optimality property.

3. In this thesis, a brute-force method was employed to utilize lookahead information in the Prospect Algorithm. Its number of computations (calculation time) exploded exponentially with the number of items known. It would be interesting to explore methods for this task that require fewer computations.

APPENDIX A

SIMULATION METHOD AND DATA DISTRIBUTIONS

A.1 Overview

This appendix is a description of the simulation methodology and data distributions used in the thesis. A simple generator for the item sizes is used for the simulations: a list of N items representing the item distribution is kept and simulation items are generated by randomly choosing items out of the list. Item distributions used are summarized in Table A.1 below. Many of the distributions are discrete approximations of normal (Gaussian) distributions, and for those the name contains the mean and standard deviation for the continuous distribution we are trying to mimic. The resulting discrete approximation will not have exactly the same standard deviation, and the resulting standard deviation with two significant digits is shown in the σ column of Table A.1. Detailed descriptions of all the distributions used are included in Sections A.2 to A.21 of this Appendix. All simulations are made with the same random number generator that is given in Section A.22, and the random seed used was 1537759668 for the item stream unless noted otherwise. If a second stream was needed, such as when simulating doubling in Chapter 8, then the random seed 0209673966 was used for that.

A.2 $N_D(100,10)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 2000 items. The item average is 100, and the standard deviation is 10.01. The histogram for the distribution is given in Table A.2.

Table A.1: List of distributions used in simulations.

Name	Description	μ	σ	min	max	skewness	kurtosis
$N_D(100,10)$	Discrete Gaussian	100.0	10.01	68	132	0	-0.055
$N_D(100,15)$	Discrete Gaussian	100.0	14.86	55	145	0	-0.126
$N_D(100,20)$	Discrete Gaussian	100.0	20.01	39	161	0	-0.064
$N_D(50,10)$	Discrete Gaussian	50.0	9.99	21	79	0	-0.192
$N_D(10,1)$	Discrete gaussian	10.0	0.99	8	12	0	-0.381
$N_D(10,1.5)$	Discrete Gaussian	10.0	1.53	6	14	0	-0.079
$N_D(10,2)$	Discrete Gaussian	10.0	1.97	5	15	0	-0.144
$U_D(75,125)$	Discrete uniform	100.0	14.87	75	125	0	-1.200
$U_D(8,12)$	Discrete uniform	10.0	1.58	8	12	0	-1.200
LS(100,10)	Left skewed	100.0	10.00	63	115	-0.645	-0.005
LS(100,15)	Left skewed	100.0	14.85	44	120	-0.853	0.342
LS(100,20)	Left skewed	100.0	20.00	33	132	-0.550	-0.187
RS(100,10)	Right skewed	100.0	10.00	85	137	0.645	-0.005
RS(100,15)	Right skewed	100.0	14.85	80	156	0.853	0.342
RS(100,20)	Right skewed	100.0	20.00	68	167	0.550	-0.187
TP(100,10)	Twin peaked	100.0	10.00	74	126	0	-1.203
TP(100,15)	Twin peaked	100.0	14.86	70	130	0	-1.733
TP(100,20)	Twin peaked	100.0	20.00	61	139	0	-1.764
Salmon	Salmon data	652.7	43.02	580	740	0.153	-1.018
Breasts	Earlybird data	326.8	38.80	217	441	0.427	0.618
Drums	Earlybird data	97.0	14.65	63	143	0.127	-0.196
Thighs	Earlybird data	204.3	33.02	123	339	0.279	0.368
Wings	Earlybird data	86.0	17.40	40	154	0.709	1.204

Table A.2: Histogram for the $N_D(100,10)$ distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
68	1	79	9	90	48	101	79	112	39	123	6
69	1	80	11	91	53	102	78	113	34	124	4
70	1	81	13	92	58	103	76	114	30	125	4
71	1	82	16	93	62	104	74	115	26	126	3
72	2	83	19	94	67	105	70	116	22	127	2
73	2	84	22	95	70	106	67	117	19	128	2
74	3	85	26	96	74	107	62	118	16	129	1
75	4	86	30	97	76	108	58	119	13	130	1
76	4	87	34	98	78	109	53	120	11	131	1
77	6	88	39	99	79	110	48	121	9	132	1
78	7	89	44	100	80	111	44	122	7		

A.3 $N_D(100,15)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 1995 items. The item average is 100, and the standard deviation is 14.86. The histogram for the distribution is given in Table A.3.

A.4 $N_D(100,20)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 2000 items. The item average is 100, and the standard deviation is 20.01. The histogram for the distribution is given in Table A.4.

A.5 $N_D(50,10)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 100 items. The item average is 50, and the standard deviation is 9.50. The histogram for the distribution is given in Table A.5.

Table A.3: Histogram for the $N_D(100,15)$ distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
55	1	71	8	86	34	101	53	116	30	131	6
56	1	72	9	87	37	102	53	117	28	132	5
57	1	73	11	88	39	103	52	118	26	133	5
58	1	74	12	89	41	104	51	119	24	134	4
59	1	75	13	90	43	105	50	120	22	135	3
60	2	76	15	91	44	106	49	121	20	136	3
61	2	77	16	92	46	107	48	122	18	137	3
62	2	78	18	93	48	108	46	123	16	138	2
63	3	79	20	94	49	109	44	124	15	139	2
64	3	80	22	95	50	110	43	125	13	140	2
65	3	81	24	96	51	111	41	126	12	141	1
66	4	82	26	97	52	112	39	127	11	142	1
67	5	83	28	98	53	113	37	128	9	143	1
68	5	84	30	99	53	114	34	129	8	144	1
69	6	85	32	100	53	115	32	130	7	145	1
70	7										

A.6 $N_D(10,1.0)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 100 items. The item average is 10, and the standard deviation is 0.99. The histogram for the distribution is given in Table A.6.

A.7 $N_D(10,1.5)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 101 items. The item average is 10, and the standard deviation is 1.53. The histogram for the distribution is given in Table A.7.

Table A.4: Histogram for the $N_D(100,20)$ distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
39	1	60	5	81	25	102	40	122	22	142	4
40	1	61	6	82	27	103	39	123	21	143	4
41	1	62	7	83	28	104	39	124	19	144	4
42	1	63	7	84	29	105	39	125	18	145	3
43	1	64	8	85	30	106	38	126	17	146	3
44	1	65	9	86	31	107	38	127	16	147	3
45	1	66	9	87	32	108	37	128	15	148	2
46	1	67	10	88	33	109	36	129	14	149	2
47	1	68	11	89	34	110	35	130	13	150	2
48	1	69	12	90	35	111	34	131	12	151	2
49	2	70	13	91	36	112	33	132	11	152	1
50	2	71	14	92	37	113	32	133	10	153	1
51	2	72	15	93	38	114	31	134	9	154	1
52	2	73	16	94	38	115	30	135	9	155	1
53	3	74	17	95	39	116	29	136	8	156	1
54	3	75	18	96	39	117	28	137	7	157	1
55	3	76	19	97	39	118	27	138	7	158	1
56	4	77	21	98	40	119	25	139	6	159	1
57	4	78	22	99	40	120	24	140	5	160	1
58	4	79	23	100	40	121	23	141	5	161	1
59	5	80	24	101	40						

A.8 $N_D(10,2)$

This distribution is a discrete approximation of a continuous Normal (Gaussian) distribution. The distribution is made out of a list of 100 items. The item average is 10, and the standard deviation is 1.97. The histogram for the distribution is given in Table A.8.

A.9 $U_D(75,125)$

This is a discrete uniform distribution between 75 and 125. The distribution is made out of a list of 51 items. The item average is 100, and the standard deviation is 14.87. The histogram for the distribution is given in Table A.9.

Table A.5: Histogram for the $N_D(50,10)$ distribution.

Item	n	Item	n	Item	n
29	1	44	3	59	3
30	1	45	4	60	2
31	1	46	4	61	2
32	1	47	4	62	2
33	1	48	4	63	2
34	1	49	4	64	1
35	1	50	4	65	1
36	1	51	4	66	1
37	2	52	4	67	1
38	2	53	4	68	1
39	2	54	4	69	1
40	2	55	4	70	1
41	3	56	3	71	1
42	3	57	3		
43	3	58	3		

Table A.6: Histogram for the $N_D(10,1.0)$ distribution.

Item	n
8	6
9	24
10	39
11	24
12	6

A.10 U_D(8,12)

This is a discrete uniform distribution between 8 and 12. The distribution is made out of a list of 5 items. The item average is 10, and the standard deviation is 1.58. The histogram for the distribution is given in Table A.10.

A.11 LS(100,10)

This is a left skewed distribution (long left tail or negative skewness). The distribution is made out of a list of 2001 items. The item average is 100, the standard deviation is 10.00, skewness is -0.645, and kurtosis -0.005. The histogram for the distribution

Table A.7: Histogram for the $N_D(10,1.5)$ distribution.

Item	n
6	1
7	4
8	11
9	21
10	27
11	21
12	11
13	4
14	1

Table A.8: Histogram for the $N_D(10,2)$ distribution.

Item	n
5	1
6	3
7	6
8	12
9	18
10	20
11	18
12	12
13	6
14	3
15	1

is given in Table A.11.

A.12 LS(100,15)

This is a left skewed distribution (long left tail or negative skewness). The distribution is made out of a list of 1995 items. The item average is 100, the standard deviation is 14.85, skewness is -0.853, and kurtosis 0.342. The histogram for the distribution is given in Table A.12.

Table A.9: Histogram for the $U_D(75,125)$ distribution.

Item	n	Item	n	Item	n	Item	n	Item	n
75	1	86	1	96	1	106	1	116	1
76	1	87	1	97	1	107	1	117	1
77	1	88	1	98	1	108	1	118	1
78	1	89	1	99	1	109	1	119	1
79	1	90	1	100	1	110	1	120	1
80	1	91	1	101	1	111	1	121	1
81	1	92	1	102	1	112	1	122	1
82	1	93	1	103	1	113	1	123	1
83	1	94	1	104	1	114	1	124	1
84	1	95	1	105	1	115	1	125	1
85	1								

Table A.10: Histogram for the $U_D(8,12)$ distribution.

Item	n
8	1
9	1
10	1
11	1
12	1

A.13 LS(100,20)

This is a left skewed distribution (long left tail or negative skewness). The distribution is made out of a list of 1998 items. The item average is 100, the standard deviation is 20.00, skewness is -0.550, and kurtosis -0.187. The histogram for the distribution is given in Table A.13.

A.14 RS(100,10)

This is a right skewed distribution (long right tail or positive skewness). The distribution is made out of a list of 2001 items. The item average is 100, the standard deviation is 10.00, skewness is 0.645, and kurtosis -0.005. The histogram for the distribution is given in Table A.14.

Table A.11: Histogram for the LS(100,10) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
63	1	72	4	81	15	90	41	99	69	108	75
64	1	73	4	82	17	91	44	100	71	109	72
65	1	74	5	83	20	92	48	101	73	110	70
66	1	75	6	84	22	93	51	102	74	111	68
67	1	76	7	85	25	94	55	103	75	112	66
68	2	77	9	86	28	95	58	104	76	113	64
69	2	78	10	87	31	96	61	105	76	114	61
70	2	79	12	88	34	97	64	106	76	115	57
71	3	80	13	89	38	98	67	107	75		

Table A.12: Histogram for the LS(100,15) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
44	1	57	3	70	9	83	22	96	41	109	56
45	1	58	3	71	10	84	24	97	43	110	57
46	1	59	3	72	10	85	25	98	44	111	57
47	1	60	3	73	11	86	26	99	45	112	58
48	1	61	4	74	12	87	28	100	46	113	58
49	1	62	4	75	13	88	29	101	48	114	58
50	1	63	5	76	14	89	31	102	49	115	58
51	1	64	5	77	15	90	32	103	50	116	58
52	1	65	6	78	16	91	34	104	52	117	58
53	2	66	6	79	17	92	35	105	53	118	58
54	2	67	7	80	18	93	37	106	54	119	58
55	2	68	7	81	20	94	38	107	55	120	58
56	2	69	8	82	21	95	40	108	55		

A.15 RS(100,15)

This is a right skewed distribution (long right tail or positive skewness). The distribution is made out of a list of 1995 items. The item average is 100, the standard deviation is 14.85, skewness is 0.853, and kurtosis 0.342. The histogram for the distribution is given in Table A.15.

Table A.13: Histogram for the LS(100,20) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
33	1	50	3	67	11	84	25	101	36	118	34
34	1	51	3	68	11	85	25	102	37	119	34
35	1	52	4	69	12	86	26	103	37	120	33
36	1	53	4	70	13	87	27	104	37	121	32
37	1	54	4	71	13	88	28	105	37	122	32
38	1	55	5	72	14	89	29	106	37	123	31
39	1	56	5	73	15	90	30	107	37	124	30
40	1	57	5	74	16	91	30	108	37	125	29
41	1	58	6	75	17	92	31	109	37	126	29
42	1	59	6	76	18	93	32	110	37	127	29
43	2	60	7	77	18	94	33	111	37	128	27
44	2	61	7	78	19	95	33	112	37	129	26
45	2	62	8	79	20	96	34	113	36	130	25
46	2	63	8	80	21	97	34	114	36	131	25
47	2	64	9	81	22	98	35	115	36	132	23
48	2	65	9	82	23	99	35	116	35		
49	3	66	10	83	24	100	36	117	35		

A.16 RS(100,20)

This is a right skewed distribution (long right tail or positive skewness). The distribution is made out of a list of 1998 items. The item average is 100, the standard deviation is 20.00, skewness is 0.550, and kurtosis -0.187. The histogram for the distribution is given in Table A.16.

A.17 TP(100,10)

This is a twin peak distribution (items in the middle are missing). The distribution is made out of a list of 1998 items. The item average is 100, the standard deviation is 10.00, skewness is 0, and kurtosis -1.203. The histogram for the distribution is given in Table A.17.

Table A.14: Histogram for the RS(100,10) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
85	57	94	76	103	64	112	34	121	12	130	2
86	61	95	76	104	61	113	31	122	10	131	2
87	64	96	76	105	58	114	28	123	9	132	2
88	66	97	75	106	55	115	25	124	7	133	1
89	68	98	74	107	51	116	22	125	6	134	1
90	70	99	73	108	48	117	20	126	5	135	1
91	72	100	71	109	44	118	17	127	4	136	1
92	75	101	69	110	41	119	15	128	4	137	1
93	75	102	67	111	38	120	13	129	3		

Table A.15: Histogram for the RS(100,15) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
80	58	93	55	106	38	119	20	132	7	145	2
81	58	94	54	107	37	120	18	133	7	146	2
82	58	95	53	108	35	121	17	134	6	147	2
83	58	96	52	109	34	122	16	135	6	148	1
84	58	97	50	110	32	123	15	136	5	149	1
85	58	98	49	111	31	124	14	137	5	150	1
86	58	99	48	112	29	125	13	138	4	151	1
87	58	100	46	113	28	126	12	139	4	152	1
88	58	101	45	114	26	127	11	140	3	153	1
89	57	102	44	115	25	128	10	141	3	154	1
90	57	103	43	116	24	129	10	142	3	155	1
91	56	104	41	117	22	130	9	143	3	156	1
92	55	105	40	118	21	131	8	144	2		

A.18 TP(100,15)

This is a twin peak distribution (items in the middle are missing). The distribution is made out of a list of 1994 items. The item average is 100, the standard deviation is 14.86, skewness is 0, and kurtosis -1.733. The histogram for the distribution is given in Table A.18.

Table A.16: Histogram for the RS(100,20) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
68	23	85	36	102	35	119	22	136	9	153	2
69	25	86	36	103	34	120	21	137	8	154	2
70	25	87	36	104	34	121	20	138	8	155	2
71	26	88	37	105	33	122	19	139	7	156	2
72	27	89	37	106	33	123	18	140	7	157	2
73	29	90	37	107	32	124	18	141	6	158	1
74	29	91	37	108	31	125	17	142	6	159	1
75	29	92	37	109	30	126	16	143	5	160	1
76	30	93	37	110	30	127	15	144	5	161	1
77	31	94	37	111	29	128	14	145	5	162	1
78	32	95	37	112	28	129	13	146	4	163	1
79	32	96	37	113	27	130	13	147	4	164	1
80	33	97	37	114	26	131	12	148	4	165	1
81	34	98	37	115	25	132	11	149	3	166	1
82	34	99	36	116	25	133	11	150	3	167	1
83	35	100	36	117	24	134	10	151	3		
84	35	101	35	118	23	135	9	152	2		

A.19 TP(100,20)

This is a twin peak distribution (items in the middle are missing). The distribution is made out of a list of 1996 items. The item average is 100, the standard deviation is 20.00, skewness is 0, and kurtosis -1.764. The histogram for the distribution is given in Table A.19.

A.20 Fresh Salmon

This distribution was logged at a Norwegian salmon farming company. The data entries are weights of whole, fresh farmed salmon and the original weights were from 2900 g to 3700 g with weighing resolution of 5 g. The data is normalized by dividing by 5 and that gave us the distribution given in Table A.20. The distribution is made out of a list of 2070 items. The item average is 652.7, and the standard deviation is 43.02. This distribution is called *FS(*653,43*)*. The histogram for the distribution is

Table A.17: Histogram for the TP(100,10) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
74	1	83	16	92	108	101	0	110	80	119	9
75	1	84	21	93	123	102	0	111	67	120	6
76	1	85	28	94	137	103	0	112	55	121	4
77	2	86	36	95	150	104	0	113	45	122	3
78	3	87	45	96	0	105	150	114	36	123	2
79	4	88	55	97	0	106	137	115	28	124	1
80	6	89	67	98	0	107	123	116	21	125	1
81	9	90	80	99	0	108	108	117	16	126	1
82	12	91	94	100	0	109	94	118	12		

Table A.18: Histogram for the TP(100,15) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
70	1	81	34	92	0	103	0	114	112	125	5
71	1	82	45	93	0	104	0	115	91	126	4
72	2	83	58	94	0	105	0	116	73	127	2
73	2	84	73	95	0	106	0	117	58	128	2
74	4	85	91	96	0	107	0	118	45	129	1
75	5	86	112	97	0	108	0	119	34	130	1
76	7	87	137	98	0	109	0	120	26		
77	10	88	163	99	0	110	0	121	19		
78	14	89	193	100	0	111	193	122	14		
79	19	90	0	101	0	112	163	123	10		
80	26	91	0	102	0	113	137	124	7		

given in Table A.20.

A.21 Earlybird Data

These distributions were logged at the South African poultry manufacturer Earlybird Farms in 1998. They consist of four sets of weights; one for each cut of a whole chicken, breasts, drums, thighs, and wings. The histograms for the distributions are given in Tables A.21 to A.24.

Table A.19: Histogram for the TP(100,20) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
61	1	75	29	89	0	103	0	117	119	131	7
62	1	76	35	90	0	104	0	118	102	132	5
63	1	77	43	91	0	105	0	119	88	133	4
64	2	78	52	92	0	106	0	120	74	134	3
65	2	79	62	93	0	107	0	121	62	135	2
66	3	80	74	94	0	108	0	122	52	136	2
67	4	81	88	95	0	109	0	123	43	137	1
68	5	82	102	96	0	110	0	124	35	138	1
69	7	83	119	97	0	111	0	125	29	139	1
70	9	84	136	98	0	112	0	126	23		
71	11	85	156	99	0	113	0	127	18		
72	15	86	0	100	0	114	0	128	15		
73	18	87	0	101	0	115	156	129	11		
74	23	88	0	102	0	116	136	130	9		

Table A.20: Histogram for the FS(653,43) distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
580	11	607	15	634	15	661	18	688	15	715	12
581	15	608	16	635	16	662	17	689	9	716	11
582	11	609	18	636	13	663	17	690	14	717	9
583	17	610	10	637	18	664	16	691	10	718	13
584	13	611	16	638	12	665	14	692	13	719	10
585	9	612	18	639	25	666	18	693	14	720	7
586	18	613	14	640	20	667	14	694	10	721	6
587	15	614	17	641	19	668	13	695	17	722	9
588	14	615	13	642	19	669	16	696	14	723	10
589	22	616	13	643	13	670	15	697	15	724	4
590	11	617	16	644	15	671	17	698	10	725	7
591	8	618	17	645	16	672	14	699	5	726	8
592	13	619	14	646	14	673	14	700	10	727	9
593	18	620	11	647	14	674	12	701	15	728	8
594	12	621	19	648	15	675	13	702	10	729	10
595	17	622	14	649	22	676	9	703	12	730	12
596	11	623	24	650	22	677	10	704	12	731	7
597	8	624	17	651	18	678	13	705	7	732	6
598	12	625	12	652	20	679	22	706	9	733	6
599	9	626	8	653	16	680	15	707	9	734	6
600	15	627	13	654	10	681	12	708	14	735	5
601	17	628	15	655	17	682	9	709	12	736	7
602	15	629	12	656	12	683	9	710	7	737	7
603	14	630	16	657	13	684	18	711	12	738	5
604	13	631	7	658	13	685	11	712	15	739	9
605	12	632	8	659	15	686	10	713	12	740	2
606	16	633	12	660	7	687	13	714	4		

Table A.21: Histogram for the Earlybird breasts distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
217	1	272	1	301	6	329	3	359	11	401	2
235	1	273	3	303	5	331	7	361	7	409	2
239	1	275	4	305	9	333	12	363	6	415	4
241	2	277	2	307	11	335	5	365	3	417	1
242	1	281	3	309	11	337	12	367	2	421	1
247	1	283	6	311	16	339	8	369	4	425	2
249	2	284	2	312	1	341	17	371	3	427	2
251	1	285	3	313	4	343	5	373	5	431	4
253	2	287	3	315	11	345	6	375	4	437	1
255	4	289	1	317	19	347	6	377	2	439	1
261	1	291	16	319	5	349	1	385	1	441	1
262	1	293	6	321	12	351	8	389	1		
265	2	295	9	323	3	353	4	391	2		
269	2	297	3	325	7	355	3	393	2		
271	1	299	5	327	7	357	7	397	3		

Table A.22: Histogram for the Earlybird drums distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
63	2	77	10	91	19	103	17	117	7	141	2
65	5	79	12	93	19	105	17	119	8	143	1
67	2	81	12	95	17	107	12	121	10		
69	4	83	29	97	24	109	21	123	2		
71	5	85	17	98	1	111	10	125	4		
73	4	87	12	99	24	113	17	127	3		
75	2	89	15	101	19	115	11	129	2		

Table A.23: Histogram for the Earlybird thighs distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
123	1	159	13	185	7	211	4	235	4	263	1
125	1	161	1	187	6	213	6	237	4	265	1
135	2	163	5	189	7	215	14	239	1	266	1
137	1	165	5	191	10	217	12	241	4	267	3
139	2	167	4	193	10	219	17	243	3	269	1
141	1	169	8	195	2	221	12	245	7	275	1
143	1	171	6	197	9	223	11	247	3	279	1
145	2	173	6	199	13	225	12	249	3	283	2
149	6	175	6	201	4	227	10	251	1	285	1
151	5	177	5	203	6	228	1	253	2	289	1
153	3	179	13	205	10	229	11	255	2	295	1
155	1	181	8	207	7	231	8	259	7	313	1
157	2	183	10	209	7	233	5	261	2	339	1

Table A.24: Histogram for the Earlybird wing distribution.

Item	n	Item	n	Item	n	Item	n	Item	n	Item	n
40	1	64	8	80	16	94	21	110	4	138	1
48	1	66	19	82	25	96	9	112	9	144	3
52	1	68	12	84	18	98	18	114	2	154	2
54	2	70	14	86	22	100	10	116	5		
56	1	72	11	88	18	102	8	118	3		
58	7	74	17	90	22	104	6	120	4		
60	6	76	10	92	12	106	2	128	3		
62	11	78	20	93	1	108	12	130	1		

A.22 Random Number Generator

The following C-code was used for random number generation. It is a slightly modified version of the RANROT B random number generator published by Agner Fog in 1998 [57]. In Table A.25 on Page 227, the first twenty items sampled using our code are listed for ten different seeds. The items are in all cases sampled from the $N_D(100,15)$ distribution. Those who want to compile our random number generation code can use this data to cross-check their program against ours.

```
//  RandGen random number generator.
//  Based on RANROT by Agner Fog 1998 (http://agner.org/random/)
//  Modified by Agni sgeirsson in 1999.

#include <time.h>
#include <stdlib.h>
#include <assert.h>

#include "RandGen.h"

//Function definitions
#define MAX(X,Y) ((X)>=(Y)?(X):(Y))
#define MIN(X,Y) ((X)<=(Y)?(X):(Y))
#define ABS(X) ((X)<=0?-(X):(X))

// define parameters
#define KK 17
#define JJ 10
#define R1 13
#define R2  5

#define NG 5

int p1[NG], p2[NG];                    // indexes into buffer
int imin[NG]={0,0,0,0,0};
int iinterval[NG]={99,99,99,99,99};  // interval for iRandom
uint32 randbuffer[NG][KK][2];        // history buffer
union  // used for conversion to float
{
  trfloat randp1;
  uint32 randbits[3];
} fr[NG];
char NotInitialized[NG]={1,1,1,1,1};


int __stdcall rg_SetRandomInterval(int H,int min, int max)
{
```

```
  // check bounds on H, and modify if necessary
  int h;
  h=MIN(NG-1,H);
  h=MAX(0,h);
  // set interval for iRandom[k]
  imin[h] = min; iinterval[h] = max - min + 1;
  return(0);
}



void step(int h)
{
  // generate next random number
  uint32 a, b;
  // generate next number
  b = _lrotr(randbuffer[h][p1[h]][0], R1) + randbuffer[h][p2[h]][0];
  a = _lrotr(randbuffer[h][p1[h]][1], R2) + randbuffer[h][p2[h]][1];
  randbuffer[h][p1[h]][0] = a;
  randbuffer[h][p1[h]][1] = b;
  // rotate list pointers
  if (--p1[h] < 0) p1[h] = KK - 1;
  if (--p2[h] < 0) p2[h] = KK - 1;
  // convert to float
  fr[h].randbits[0] = a;
  // 64 bits floats = 52 bits resolution
  fr[h].randbits[1] = (b & 0x000FFFFF) | 0x3FF00000;
}

int __stdcall rg_RandomInit(int H,uint32 seed)
{
  // this function initializes the random number generator.
  int i, j ,h;
  // check bounds on H, and modify if necessary
  h=MIN(NG-1,H);
  h=MAX(0,h);
  // make sure seed != 0
  if (seed==0) seed = -1;

  // make random numbers and put them into the buffer
  for (i=0; i<KK; i++)
  {
    for (j=0; j<2; j++)
    {
      seed ^= seed << 13;
```

```
      seed ^= seed >> 17;
      seed ^= seed << 5;
      randbuffer[h][i][j] = seed;
    }
  }
  // set exponent of randp1
  fr[h].randp1 = 1.5;
  // check that IEEE double precision float format used
  assert(fr[h].randbits[1]==0x3FF80000);
  // check that 32 bits integers used
  assert(sizeof(uint32)==4);
  // initialize pointers to circular buffer
  p1[h] = 0;  p2[h] = JJ;
  // randomize some more
  for (i=0; i<97; i++) step(h);
  NotInitialized[h]=0;
  return(0);
}

trfloat __stdcall rg_Random(int H)
{
  trfloat r;
  // check bounds on H, and modify if necessary
  int h;
  h=MIN(NG-1,H);
  h=MAX(0,h);
  if(NotInitialized[h]) rg_RandomInit(h,time(0));
  // returns a random number between 0 and 1.
  r = fr[h].randp1 - 1.;
  step(h);
  return r;
}


int __stdcall rg_iRandom(int H)
{
  int i,h;
  // check bounds on H, and modify if necessary
  h=MIN(NG-1,H);
  h=MAX(0,h);
  // get integer random number
  i = (int)(iinterval[h] * rg_Random(h));
  if (i >= iinterval[h]) i = iinterval[h];
  return imin[h] + i;
}
```

Table A.25: Cross checking data for random number generation. The items are drawn from the $N_D(100,15)$ distribution described in Appendix A.3.

Data no.	Seed	First 20 items
1	1537759668	94 108 93 116 117 94 92 102 83 107 82 112 88 105 110 83 119 117 107 78
2	844405318	111 89 81 96 89 90 82 111 118 72 104 110 87 106 101 110 102 126 118 104
3	752861129	73 101 97 114 117 83 104 111 97 95 75 95 102 113 90 107 110 120 101 73
4	1946276008	92 122 108 100 99 93 119 98 100 108 119 77 84 98 106 93 95 102 81 120
5	777694791	107 100 89 94 91 119 98 102 84 106 101 89 97 123 92 126 78 116 82 100
6	170651580	83 95 97 99 104 115 82 105 118 100 110 65 84 103 103 88 68 120 116 81
7	691114543	79 106 90 101 81 104 123 143 108 122 110 73 118 75 81 88 86 100 93 88
8	756107051	106 93 111 124 110 112 95 98 86 130 105 99 119 103 84 69 89 116 88 103
9	972909786	90 139 107 93 73 96 126 99 104 83 126 128 95 120 89 104 133 108 109 86
10	203208819	92 111 110 95 87 112 105 107 99 84 91 95 113 100 115 112 102 107 113 90

APPENDIX B

DETAILED COMPARISON OF THE PR AND PD ALGORITHMS FOR BIN-COVERING

In this appendix the full comparison analysis of the average overfill performance of the two versions of the Prospect Algorithm for Bin-Covering is provided. These are the Prospect Ratio algorithm for Bin-Covering (PR) and the Prospect Differential algorithm for Bin-Covering (PD). Tables B.1 and B.2 list the 95% confidence intervals for the simulations from Figures 3.9 to 3.12.

Table B.1: Comparison of 95% confidence intervals for the average overfill obtained by the PR and PD Algorithms for the $N_D(100,10)$ and $N_D(100,15)$ distributions.

Bin	$N_D(100,10)$ - Figure 3.9			$N_D(100,15)$ - Figure 3.10		
capacity	PR	PD	Winner	PR	PD	Winner
200	[8.53,8.67]	[8.77,8.90]	PR	[8.62,8.75]	[8.90,9.02]	PR
220	[44.2,44.4]	[43.5,43.6]	PD	[26.3,26.5]	[25.9,26.1]	PD
240	[54.6,54.7]	[54.6,54.7]	TIE	[40.1,40.2]	[39.6,39.7]	PD
260	[39.9,40.0]	[39.9,40.0]	TIE	[37.0,37.1]	[36.7,36.9]	PD
280	[19.9,20.0]	[19.9,20.0]	TIE	[19.8,19.9]	[19.8,19.9]	TIE
300	[5.99,6.15]	[6.01,6.15]	TIE	[4.82,4.88]	[5.16,5.22]	PR
320	[27.2,27.4]	[28.7,28.8]	PR	[12.4,12.6]	[12.1,12.3]	PD
340	[42.0,42.2]	[45.5,45.6]	PR	[20.3,20.5]	[20.5,20.7]	PR
360	[37.2,37.3]	[37.9,38.0]	PR	[22.2,22.4]	[23.5,23.7]	PR
380	[19.9,20.0]	[19.9,20.0]	TIE	[14.6,14.8]	[14.5,14.7]	TIE
400	[4.70,4.82]	[4.82,4.91]	PR	[3.53,3.60]	[3.76,3.84]	PR
420	[14.6,14.8]	[21.4,21.5]	PR	[3.28,3.36]	[2.97,3.04]	PD
440	[24.9,25.0]	[28.0,28.2]	PR	[4.12,4.20]	[4.10,4.20]	TIE
460	[29.7,29.8]	[30.0,30.1]	PR	[5.05,5.19]	[5.08,5.20]	TIE
480	[18.7,18.8]	[18.5,18.6]	PD	[4.99,5.10]	[4.25,4.33]	PD
500	[3.59,3.70]	[3.98,4.07]	PR	[2.50,2.55]	[2.44,2.48]	PD
520	[8.98,9.05]	[15.4,15.6]	PR	[1.73,1.76]	[1.72,1.74]	TIE
540	[13.6,13.8]	[14.8,14.9]	PR	[1.62,1.64]	[1.61,1.63]	TIE
560	[18.5,18.6]	[18.7,18.8]	PR	[1.63,1.65]	[1.59,1.61]	PD
580	[15.0,15.2]	[14.6,14.7]	PD	[1.70,1.72]	[1.63,1.65]	PD
600	[2.61,2.70]	[3.36,3.44]	PR	[1.66,1.68]	[1.56,1.60]	PD
620	[4.71,4.86]	[4.27,4.47]	PD	[1.32,1.35]	[1.33,1.35]	TIE
640	[4.41,4.57]	[7.22,7.36]	PR	[1.24,1.27]	[1.24,1.26]	TIE
660	[6.19,6.36]	[9.23,9.39]	PR	[1.21,1.23]	[1.17,1.19]	PD
680	[8.53,8.69]	[8.28,8.43]	PD	[1.17,1.19]	[1.13,1.15]	PD
700	[1.83,1.91]	[2.17,2.27]	PR	[1.15,1.17]	[1.14,1.15]	TIE
720	[1.17,1.24]	[1.04,1.06]	PD	[1.09,1.11]	[1.08,1.10]	TIE
740	[0.80,0.85]	[0.85,0.91]	PR	[1.09,1.10]	[1.08,1.09]	TIE
760	[0.98,1.05]	[1.16,1.21]	PR	[1.07,1.09]	[1.07,1.08]	TIE
780	[2.06,2.12]	[1.61,1.71]	PD	[1.05,1.06]	[1.05,1.06]	TIE
800	[1.23,1.27]	[1.14,1.19]	PD	[1.05,1.06]	[1.03,1.06]	TIE

Table B.2: Comparison of 95% confidence intervals for the average overfill obtained by the PR and PD Algorithms for the $N_D(100,20)$ and $U_D(75,125)$ distributions.

Bin	$N_D(100,20)$ - Figure 3.11			U{75-125} - Figure3.12		
capacity	PR	PD	Winner	PR	PD	Winner
200	[10.4,10.5]	[10.6,10.7]	PR	[8.93,9.04]	[8.93,9.04]	TIE
220	[19.4,19.6]	[19.9,20.0]	PR	[17.4,17.5]	[17.3,17.5]	TIE
240	[26.6,26.8]	[26.3,26.4]	PD	[34.8,34.9]	[34.5,34.7]	PD
260	[27.6,27.8]	[28.0,28.2]	PR	[39.9,40.0]	[39.9,40.0]	TIE
280	[17.1,17.3]	[17.9,18.1]	PR	[19.9,20.0]	[19.9,20.0]	TIE
300	[5.92,6.02]	[6.35,6.45]	PR	[5.72,5.83]	[5.67,5.78]	TIE
320	[4.77,4.85]	[4.62,4.69]	PD	[7.59,7.69]	[7.23,7.34]	PD
340	[5.37,5.47]	[5.30,5.41]	TIE	[12.0,12.1]	[11.8,11.9]	PD
360	[5.91,6.08]	[5.89,6.03]	TIE	[21.0,21.2]	[19.9,20.1]	PD
380	[5.21,5.31]	[5.05,5.15]	PD	[19.9,20.0]	[19.9,20.0]	TIE
400	[3.67,3.74]	[3.68,3.73]	TIE	[3.64,3.70]	[3.49,3.55]	PD
420	[3.01,3.05]	[3.02,3.06]	TIE	[2.78,2.83]	[2.47,2.49]	PD
440	[2.82,2.85]	[2.81,2.83]	TIE	[2.29,2.32]	[2.26,2.29]	PD
460	[2.64,2.69]	[2.64,2.67]	TIE	[2.42,2.45]	[2.37,2.41]	PD
480	[2.53,2.57]	[2.51,2.55]	TIE	[4.92,5.00]	[3.57,3.67]	PD
500	[2.42,2.46]	[2.43,2.46]	TIE	[3.05,3.13]	[3.02,3.09]	TIE
520	[2.28,2.31]	[2.30,2.33]	TIE	[1.90,1.93]	[1.80,1.82]	PD
540	[2.22,2.24]	[2.20,2.23]	TIE	[1.74,1.76]	[1.66,1.69]	PD
560	[2.11,2.15]	[2.10,2.14]	TIE	[1.68,1.70]	[1.62,1.63]	PD
580	[1.97,2.00]	[1.96,1.99]	TIE	[1.72,1.74]	[1.64,1.65]	PD
600	[1.87,1.91]	[1.87,1.90]	TIE	[1.83,1.85]	[1.70,1.72]	PD
620	[1.78,1.81]	[1.78,1.81]	TIE	[1.54,1.58]	[1.43,1.46]	PD
640	[1.74,1.76]	[1.72,1.74]	TIE	[1.42,1.45]	[1.31,1.34]	PD
660	[1.68,1.70]	[1.68,1.71]	TIE	[1.30,1.33]	[1.23,1.25]	PD
680	[1.64,1.66]	[1.64,1.66]	TIE	[1.27,1.29]	[1.20,1.22]	PD
700	[1.60,1.62]	[1.59,1.61]	TIE	[1.27,1.29]	[1.19,1.21]	PD
720	[1.52,1.56]	[1.52,1.55]	TIE	[1.18,1.20]	[1.13,1.15]	PD
740	[1.43,1.46]	[1.43,1.47]	TIE	[1.15,1.17]	[1.12,1.14]	PD
760	[1.38,1.41]	[1.38,1.41]	TIE	[1.11,1.13]	[1.08,1.09]	PD
780	[1.30,1.34]	[1.32,1.35]	TIE	[1.08,1.10]	[1.06,1.07]	PD
800	[1.25,1.27]	[1.24,1.27]	TIE	[1.07,1.09]	[1.06,1.07]	PD

APPENDIX C

DETAILED COMPARISON OF THE PR AND PR+ ALGORITHMS FOR BIN-COVERING

In this appendix the full comparison analysis of the average overfill performance of the Prospect and Prospect+ Algorithms. The tables list the 95% confidence intervals for the simulations from Figures 5.6 to 5.9.

Table C.1: Comparison of 95% confidence intervals for the average overfill obtained by the PR and PR+ Algorithms for the $N_D(100,10)$ and $N_D(100,15)$ distributions.

Bin	$N_D(100,10)$ - Figure 5.6			$N_D(100,15)$ - Figure 5.7		
capacity	PR	PR+	Winner	PR	PR+	Winner
200	[8.53,8.67]	[8.82,8.96]	PR	[8.62,8.75]	[8.61,8.74]	TIE
220	[44.2,44.4]	[44.0,44.2]	TIE	[26.3,26.5]	[26.2,26.4]	TIE
240	[54.6,54.7]	[54.8,54.9]	PR	[40.1,40.2]	[40.4,40.5]	PR
260	[39.9,40.0]	[39.9,40.0]	TIE	[37.0,37.1]	[36.1,36.3]	PR+
280	[19.9,20.0]	[19.9,20.0]	TIE	[19.8,19.9]	[19.6,19.8]	TIE
300	[5.99,6.15]	[5.76,5.89]	PR+	[4.82,4.88]	[4.40,4.45]	PR+
320	[27.2,27.4]	[28.7,28.9]	PR	[12.4,12.6]	[12.1,12.2]	PR+
340	[42.0,42.2]	[42.2,42.4]	PR	[20.3,20.5]	[21.1,21.2]	PR
360	[37.2,37.3]	[36.6,36.7]	PR+	[22.2,22.4]	[21.9,22.0]	PR+
380	[19.9,20.0]	[19.7,19.8]	PR+	[14.6,14.8]	[13.9,14.0]	PR+
400	[4.70,4.82]	[4.60,4.76]	TIE	[3.53,3.60]	[3.33,3.40]	PR+
420	[14.6,14.8]	[15.5,15.6]	PR	[3.28,3.36]	[3.17,3.24]	PR+
440	[24.9,25.0]	[27.1,27.2]	PR	[4.12,4.20]	[3.98,4.12]	PR+
460	[29.7,29.8]	[29.5,29.6]	PR+	[5.05,5.19]	[5.52,5.68]	PR
480	[18.7,18.8]	[18.0,18.1]	PR+	[4.99,5.10]	[4.86,4.96]	PR+
500	[3.59,3.70]	[3.62,3.72]	TIE	[2.50,2.55]	[2.24,2.30]	PR+
520	[8.98,9.05]	[9.66,9.75]	PR	[1.73,1.76]	[1.62,1.64]	PR+
540	[13.6,13.8]	[15.9,16.0]	PR	[1.62,1.64]	[1.48,1.51]	PR+
560	[18.5,18.6]	[20.5,20.6]	PR	[1.63,1.65]	[1.38,1.42]	PR+
580	[15.0,15.2]	[14.1,14.3]	PR+	[1.70,1.72]	[1.51,1.56]	PR+
600	[2.61,2.70]	[2.56,2.64]	TIE	[1.66,1.68]	[1.34,1.36]	PR+
620	[4.71,4.86]	[4.74,4.87]	TIE	[1.32,1.35]	[1.13,1.15]	PR+
640	[4.41,4.57]	[4.60,4.79]	PR	[1.24,1.27]	[1.10,1.11]	PR+
660	[6.19,6.36]	[7.18,7.39]	PR	[1.21,1.23]	[1.06,1.08]	PR+
680	[8.53,8.69]	[7.26,7.43]	PR+	[1.17,1.19]	[1.05,1.06]	PR+
700	[1.83,1.91]	[1.68,1.76]	PR+	[1.15,1.17]	[1.04,1.05]	PR+
720	[1.17,1.24]	[1.12,1.17]	PR+	[1.09,1.11]	[0.94,0.98]	PR+
740	[0.80,0.85]	[0.63,0.66]	PR+	[1.09,1.10]	[0.88,0.91]	PR+
760	[0.98,1.05]	[0.76,0.81]	PR+	[1.07,1.09]	[0.79,0.82]	PR+
780	[2.06,2.12]	[1.76,1.87]	PR+	[1.05,1.06]	[0.70,0.73]	PR+
800	[1.23,1.27]	[0.99,1.07]	PR+	[1.05,1.06]	[0.68,0.70]	PR+

Table C.2: Comparison of 95% confidence intervals for the average overfill obtained by the PR and PR+ Algorithms for the $N_D(100,20)$ and $U_D(75,125)$ distributions.

Bin	$N_D(100,20)$ - Figure 3.11			U{75-125} - Figure3.12		
capacity	PR	PR+	Winner	PR	PR+	Winner
200	[10.4,10.5]	[10.3,10.4]	TIE	[8.93,9.04]	[8.99,9.09]	TIE
220	[19.4,19.6]	[20.2,20.4]	PR	[17.4,17.5]	[17.1,17.2]	PR+
240	[26.6,26.8]	[27.2,27.4]	PR	[34.8,34.9]	[34.9,35.0]	PR
260	[27.6,27.8]	[26.8,26.9]	PR+	[39.9,40.0]	[39.9,40.0]	TIE
280	[17.1,17.3]	[16.4,16.6]	PR+	[19.9,20.0]	[19.9,20.0]	TIE
300	[5.92,6.02]	[5.44,5.53]	PR+	[5.72,5.83]	[4.69,4.74]	PR+
320	[4.77,4.85]	[4.43,4.51]	PR+	[7.59,7.69]	[7.05,7.15]	PR+
340	[5.37,5.47]	[5.76,5.91]	PR	[12.0,12.1]	[12.0,12.1]	TIE
360	[5.91,6.08]	[6.87,7.02]	PR	[21.0,21.2]	[20.2,20.3]	PR+
380	[5.21,5.31]	[5.30,5.40]	TIE	[19.9,20.0]	[19.9,20.0]	TIE
400	[3.67,3.74]	[3.38,3.44]	PR+	[3.64,3.70]	[3.18,3.24]	PR+
420	[3.01,3.05]	[2.72,2.77]	PR+	[2.78,2.83]	[2.42,2.45]	PR+
440	[2.82,2.85]	[2.48,2.53]	PR+	[2.29,2.32]	[2.12,2.15]	PR+
460	[2.64,2.69]	[2.34,2.37]	PR+	[2.42,2.45]	[2.38,2.43]	TIE
480	[2.53,2.57]	[2.27,2.31]	PR+	[4.92,5.00]	[4.52,4.63]	PR+
500	[2.42,2.46]	[2.19,2.21]	PR+	[3.05,3.13]	[2.89,2.97]	PR+
520	[2.28,2.31]	[1.98,2.02]	PR+	[1.90,1.93]	[1.65,1.67]	PR+
540	[2.22,2.24]	[1.84,1.87]	PR+	[1.74,1.76]	[1.65,1.67]	PR+
560	[2.11,2.15]	[1.74,1.77]	PR+	[1.68,1.70]	[1.55,1.58]	PR+
580	[1.97,2.00]	[1.66,1.68]	PR+	[1.72,1.74]	[1.52,1.56]	PR+
600	[1.87,1.91]	[1.62,1.63]	PR+	[1.83,1.85]	[1.55,1.59]	PR+
620	[1.78,1.81]	[1.59,1.60]	PR+	[1.54,1.58]	[1.25,1.27]	PR+
640	[1.74,1.76]	[1.49,1.52]	PR+	[1.42,1.45]	[1.27,1.28]	PR+
660	[1.68,1.70]	[1.39,1.42]	PR+	[1.30,1.33]	[1.20,1.21]	PR+
680	[1.64,1.66]	[1.27,1.29]	PR+	[1.27,1.29]	[1.12,1.13]	PR+
700	[1.60,1.62]	[1.20,1.23]	PR+	[1.27,1.29]	[1.10,1.12]	PR+
720	[1.52,1.56]	[1.17,1.19]	PR+	[1.18,1.20]	[1.07,1.08]	PR+
740	[1.43,1.46]	[1.13,1.15]	PR+	[1.15,1.17]	[1.08,1.09]	PR+
760	[1.38,1.41]	[1.10,1.12]	PR+	[1.11,1.13]	[1.06,1.07]	PR+
780	[1.30,1.34]	[1.08,1.09]	PR+	[1.08,1.10]	[1.02,1.03]	PR+
800	[1.25,1.27]	[1.05,1.06]	PR+	[1.07,1.09]	[0.93,0.96]	PR+

APPENDIX D

CONFIDENCE INTERVALS FOR SIMULATIONS FROM CHAPTER 5

In this appendix the confidence intervals for the simulation data from Figures 5.2–5.5 are listed.

Table D.1: Comparison of 95% confidence intervals for data from Figure 5.2, the average number of open bins of the SS', Prospect-SS', and Count Prospect Algorithm. The $N_D(100,10)$ distribution is being used. Upper half shows confidence intervals for the average number of open bins, and the lower half the confidence interval for the slope of linear regression of the series. The confidence interval of the slope is boldfaced if the slope is 95% significant.

Bin	Average open bins			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[15.9,17.7]	[22.9,26.6]	[46.5,55.0]	[48.2,49.6]
620	[35.5,37.1]	[48.1,49.6]	[91.1,91.8]	[76.9,77.8]
640	[110.7,113.0]	[111.8,114.6]	[63.8,66.4]	[126.2,128.2]
660	[111.1,112.8]	[149.5,153.1]	[98.1,101.0]	[155.6,158.9]
680	[66.6,69.4]	[88.8,90.1]	[125.6,126.4]	[108.3,109.4]
700	[10.4,10.8]	[15.5,16.7]	[39.9,47.4]	[47.3,48.1]
720	[14.8,15.3]	[20.2,20.7]	[37.4,40.2]	[61.6,62.4]
740	[21.2,22.0]	[35.2,36.3]	[16.7,17.0]	[85.3,86.0]
760	[26.4,27.3]	[48.0,50.2]	[18.0,18.3]	[93.4,94.1]
780	[18.8,19.4]	[29.1,30.5]	[76.9,79.5]	[82.6,83.4]
800	[8.01,8.12]	[11.6,11.8]	[29.9,35.8]	[46.3,46.9]

Bin	Slope of linear regression			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[-0.021,0.043]	[-0.047,0.083]	[-0.017,0.033]	[-0.017,0.033]
620	[-0.038,0.019]	[-0.050,0.002]	[-0.027,0.006]	[-0.027,0.006]
640	[-0.079,0.003]	[-0.085,0.014]	[-0.061,0.005]	[-0.061,0.005]
660	[-0.024,0.035]	[-0.087,0.039]	[-0.043,0.075]	[-0.043,0.075]
680	[-0.056,0.042]	[-0.024,0.022]	[-0.022,0.018]	[-0.022,0.018]
700	**[0.001,0.015]**	[-0.001,0.040]	[-0.008,0.020]	[-0.008,0.020]
720	[-0.011,0.006]	[-0.015,0.003]	[-0.019,0.006]	[-0.019,0.006]
740	[-0.024,0.004]	[-0.025,0.012]	[-0.024,0.001]	[-0.024,0.001]
760	[-0.022,0.008]	[-0.052,0.025]	[-0.021,0.004]	[-0.021,0.004]
780	[-0.010,0.010]	[-0.020,0.028]	[-0.014,0.015]	[-0.014,0.015]
800	[-0.0010,0.003]	**[0.0003,0.008]**	[-0.0001,0.020]	[-0.0001,0.020]

Table D.2: Comparison of 95% confidence intervals for data from Figure 5.3, the average number of open bins of the SS', Prospect-SS', and Count Prospect Algorithm. The $N_D(100,15)$ distribution is being used. The Upper half shows confidence intervals for the average number of open bins, and the lower half the confidence interval for the slope of linear regression of the series. The confidence interval of the slope is boldfaced if the slope is 95% significant.

Bin	Average open bins			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[14.0,14.2]	[17.8,18.1]	[12.8,12.9]	[32.1,32.4]
620	[13.8,14.0]	[17.8,18.1]	[12.6,12.7]	[32.4,32.7]
640	[15.0,15.2]	[20.2,20.4]	[13.4,13.5]	[38.2,38.6]
660	[15.2,15.4]	[20.9,21.2]	[13.6,13.7]	[41.1,41.5]
680	[14.2,14.4]	[18.6,18.8]	[12.2,12.4]	[37.3,37.7]
700	[12.2,12.4]	[15.8,16.0]	[11.1,11.2]	[29.7,30.0]
720	[11.9,12.0]	[15.6,15.8]	[10.7,10.8]	[29.3,29.5]
740	[12.2,12.3]	[16.7,16.9]	[11.1,11.2]	[32.3,32.6]
760	[12.2,12.4]	[16.9,17.1]	[11.1,11.2]	[33.9,34.2]
780	[11.7,11.9]	[15.7,15.9]	[10.4,10.5]	[31.6,31.9]
800	[10.8,10.9]	[14.2,14.4]	[9.78,9.86]	[27.6,27.9]

Bin	Slope of linear regression			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[-0.003,0.004]	[-0.002,0.006]	[-0.004,0.009]	[-0.004,0.009]
620	[-0.004,0.002]	[-0.004,0.004]	[-0.005,0.006]	[-0.005,0.006]
640	[-0.002,0.005]	[-0.005,0.004]	[-0.010,0.004]	[-0.010,0.004]
660	[-0.006,0.002]	[-0.005,0.005]	[-0.010,0.005]	[-0.010,0.005]
680	[-0.005,0.002]	[-0.003,0.005]	[-0.010,0.005]	[-0.010,0.005]
700	[-0.004,0.002]	[-0.004,0.003]	[-0.006,0.005]	[-0.006,0.005]
720	[-0.002,0.003]	[-0.004,0.004]	[-0.007,0.003]	[-0.007,0.003]
740	[-0.003,0.003]	[-0.005,0.003]	[-0.007,0.004]	[-0.007,0.004]
760	[-0.003,0.003]	**[-0.007,-0.0002]**	[-0.006,0.005]	[-0.006,0.005]
780	[-0.004,0.001]	[-0.003,0.003]	[-0.005,0.005]	[-0.005,0.005]
800	**[0.0001,0.004]**	[-0.004,0.003]	[-0.005,0.004]	[-0.005,0.004]

Table D.3: Comparison of 95% confidence intervals for data from Figure 5.4, the average number of open bins of the SS', Prospect-SS', and Count Prospect Algorithm. The $N_D(100,20)$ distribution is being used. The Upper half shows confidence intervals for the average number of open bins, and the lower half the confidence interval for the slope of linear regression of the series. The confidence interval of the slope is boldfaced if the slope is 95% significant.

Bin	Average open bins			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[16.7,16.9]	[19.6,19.8]	[15.0,15.2]	[37.3,37.6]
620	[15.8,16.0]	[18.4,18.6]	[14.4,14.5]	[36.2,36.4]
640	[15.5,15.7]	[18.4,18.6]	[14.0,14.2]	[36.0,36.2]
660	[15.4,15.5]	[18.3,18.5]	[13.8,13.9]	[36.0,36.2]
680	[14.9,15.1]	[17.8,18.0]	[13.4,13.5]	[35.5,35.7]
700	[14.3,14.4]	[16.9,17.0]	[12.9,13.0]	[34.6,34.7]
720	[13.7,13.9]	[16.3,16.5]	[12.5,12.6]	[33.7,33.9]
740	[13.5,13.6]	[16.0,16.2]	[12.2,12.3]	[33.4,33.6]
760	[13.2,13.3]	[15.9,16.0]	[12.0,12.1]	[33.2,33.4]
780	[12.9,13.0]	[15.4,15.5]	[11.6,11.8]	[32.9,33.1]
800	[12.5,12.6]	[14.9,15.1]	[11.4,11.5]	[32.3,32.5]

Bin	Slope of linear regression			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[-0.003,0.003]	[-0.001,0.005]	[-0.005,0.004]	[-0.005,0.004]
620	[-0.002,0.004]	[-0.002,0.005]	[-0.001,0.006]	[-0.001,0.006]
640	[-0.002,0.004]	[-0.003,0.005]	[-0.002,0.006]	[-0.002,0.006]
660	[-0.002,0.004]	[-0.002,0.004]	[-0.002,0.005]	[-0.002,0.005]
680	[-0.003,0.003]	[-0.003,0.004]	[-0.002,0.004]	[-0.002,0.004]
700	[-0.002,0.003]	[-0.004,0.002]	[-0.003,0.004]	[-0.003,0.004]
720	[-0.003,0.002]	[-0.003,0.004]	[-0.003,0.004]	[-0.003,0.004]
740	[-0.002,0.002]	[-0.006,0.0008]	[-0.004,0.003]	[-0.004,0.003]
760	[-0.003,0.002]	[-0.003,0.003]	[-0.002,0.004]	[-0.002,0.004]
780	[-0.003,0.002]	[-0.004,0.002]	[-0.004,0.002]	[-0.004,0.002]
800	[-0.002,0.002]	[-0.003,0.004]	[-0.003,0.003]	[-0.003,0.003]

Table D.4: Comparison of 95% confidence intervals for data from Figure 5.5, the average number of open bins of the SS', Prospect-SS', and Count Prospect Algorithm. The $U_D(75,125)$ distribution is being used. The Upper half shows confidence intervals for the average number of open bins, and the lower half the confidence interval for the slope of linear regression of the series. The confidence interval of the slope is boldfaced if the slope is 95% significant.

Bin	Average open bins			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[10.7,10.8]	[10.7,10.8]	[11.4,11.5]	[11.6,11.8]
620	[10.3,10.4]	[9.80,9.87]	[10.7,10.8]	[10.2,10.3]
640	[10.1,10.2]	[9.67,9.74]	[10.6,10.7]	[10.2,10.3]
660	[9.83,9.92]	[9.62,9.69]	[10.4,10.5]	[10.5,10.6]
680	[9.52,9.60]	[9.36,9.44]	[9.98,10.1]	[10.3,10.5]
700	[9.14,9.23]	[8.95,9.02]	[9.49,9.57]	[9.52,9.63]
720	[8.91,8.98]	[8.57,8.64]	[9.23,9.31]	[8.93,9.01]
740	[8.70,8.77]	[8.43,8.50]	[9.09,9.15]	[8.81,8.88]
760	[8.54,8.61]	[8.28,8.34]	[8.86,8.93]	[8.79,8.86]
780	[8.28,8.34]	[8.08,8.14]	[8.59,8.65]	[8.68,8.76]
800	[8.07,8.13]	[7.83,7.89]	[8.32,8.38]	[8.26,8.33]

Bin	Slope of linear regression			
capacity	CPR	SS'/P½	SS'/P	SS'
600	[-0.002,0.002]	[-0.003,0.001]	[-0.002,0.004]	[-0.002,0.004]
620	[-0.0008,0.002]	[-0.001,0.001]	[-0.001,0.002]	[-0.001,0.002]
640	[-0.002,0.001]	[-0.002,0.0007]	[-0.001,0.002]	[-0.001,0.002]
660	[-0.0009,0.002]	[-0.0007,0.002]	[-0.001,0.003]	[-0.001,0.003]
680	[-0.001,0.001]	[-0.002,0.0007]	[-0.002,0.002]	[-0.002,0.002]
700	[-0.0009,0.002]	[-0.001,0.001]	[-0.002,0.002]	[-0.002,0.002]
720	[-0.002,0.0010]	**[0.0003,0.003]**	[-0.002,0.001]	[-0.002,0.001]
740	[-0.0001,0.002]	[-0.001,0.001]	**[-0.003,-0.0001]**	**[-0.003,-0.0001]**
760	[-0.001,0.001]	[-0.0005,0.002]	[-0.001,0.001]	[-0.001,0.001]
780	[-0.0003,0.002]	[-0.001,0.0009]	[-0.001,0.002]	[-0.001,0.002]
800	[-0.001,0.0009]	[-0.001,0.0010]	[-0.002,0.0009]	[-0.002,0.0009]

APPENDIX E

CONFIDENCE INTERVALS FOR SIMULATIONS IN CHAPTER 7

In this appendix, the confidence intervals for the simulation data in Figure 7.3 are provided.

Table E.1: Comparison of 95% confidence intervals for the data in Figure 7.3, the average bin overfill for the SSNF and SSP algorithms, with 8 weighing buffers and the $N_D(100,15)$ distribution.

Bin	SSNF	SSP	Bin	SSNF	SSP
200	[6.01,6.05]	[6.01,6.05]	520	[3.84,3.87]	[2.69,2.71]
220	[22.28,22.36]	[22.28,22.36]	540	[5.49,5.52]	[3.63,3.65]
240	[36.10,36.18]	[36.10,36.18]	560	[6.26,6.29]	[5.10,5.13]
260	[33.77,33.84]	[33.77,33.84]	580	[5.52,5.55]	[5.16,5.19]
280	[19.19,19.26]	[19.19,19.26]	600	[4.04,4.06]	[3.37,3.40]
300	[3.00,3.03]	[3.00,3.03]	620	[3.95,3.97]	[2.36,2.37]
320	[7.66,7.72]	[7.66,7.72]	640	[4.91,4.93]	[2.59,2.61]
340	[15.01,15.07]	[15.01,15.07]	660	[5.43,5.46]	[3.60,3.62]
360	[17.39,17.45]	[17.39,17.45]	680	[5.06,5.09]	[4.22,4.25]
380	[12.14,12.19]	[12.14,12.19]	700	[4.27,4.29]	[3.34,3.36]
400	[3.32,3.35]	[3.32,3.35]	720	[4.08,4.11]	[2.38,2.39]
420	[4.03,4.06]	[4.01,4.04]	740	[4.61,4.64]	[2.31,2.32]
440	[6.91,6.94]	[6.82,6.85]	760	[4.98,5.01]	[2.93,2.95]
460	[8.15,8.19]	[7.95,7.99]	780	[4.81,4.85]	[3.58,3.60]
480	[6.55,6.59]	[6.32,6.35]	800	[4.36,4.39]	[3.24,3.26]
500	[3.72,3.75]	[3.29,3.32]			

APPENDIX F

DERIVATION OF EQUATIONS (8.6), (8.7), AND (8.9)

In this appendix, the formulas for the number of loops needed to calculate the Prospect according to the three different algorithms presented in Section 8.3 are derived. However the number of operations needed to do each loop is not analyzed. The reason is that the actual work performed at each loop is not easily comparable due to the complexity of modern computers, and is outside of the scope of this thesis. Our experience strongly indicates, however, that the number of loops is the dominant factor in the speed of the calculations being analyzed here.

Note that for compactness, the maximum and minimum item sizes in the FIFO queue are denoted here as $\overline{f}$ and $\underline{f}$ instead of $f_{\min}$ and $f_{\max}$ as in the previous sections. Also, in the derivations, C is the number of convolutions kept, $R_f = \overline{f} - \underline{f}$ is the item range, and B is the length of the Prospect function. The derivations utilize the following known formulas for finite sums:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \Longrightarrow \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}; \tag{F.1}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \Longrightarrow \sum_{i=1}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6}. \tag{F.2}$$

F.1 Calculating Prospect with Program 8.3.1

Assume, without loss of relevance, that the bin size $b > 1$, making the length of the Prospect function $B > \overline{f}$ (according to Equation (8.4)). The number of loops needed

to calculate the prospect with Program 8.3.1 is given by:

$$
\begin{aligned}
&\sum_{y_1=\underline{f}}^{B}\sum_{y_2=\underline{f}}^{\min(y_1,\overline{f})} 1 \\
&= \sum_{y_1=\underline{f}}^{\overline{f}}\sum_{y_2=\underline{f}}^{y_1} 1 + \sum_{y_1=\overline{f}+1}^{B}\sum_{y_2=\underline{f}}^{\overline{f}} 1 \quad \text{(Split the sum to eliminate the min function.)} \\
&= \sum_{y_1=\underline{f}}^{\overline{f}}(y_1-\underline{f}+1) + \sum_{y_1=\overline{f}+1}^{B}(\overline{f}-\underline{f}+1) \\
&= \sum_{\hat{y}_1=1}^{\overline{f}-\underline{f}+1}\hat{y}_1 + (B-\overline{f})(\overline{f}-\underline{f}+1) \quad \text{(Change of variable, } \hat{y}_1 \equiv y_1-\underline{f}+1.\text{)} \\
&= \frac{(\overline{f}-\underline{f}+1)(\overline{f}-\underline{f}+2)}{2} + (B-\overline{f})(\overline{f}-\underline{f}+1) \text{ (Use Equation (F.1).)} \\
&= (\overline{f}-\underline{f}+1)\left(B+1-\underline{f}+\frac{\overline{f}-\underline{f}}{2}\right) \\
&= (R_f+1)\left(B+1-\underline{f}+\frac{R_f}{2}\right). \qquad \text{(F.3)}
\end{aligned}
$$

F.2 Calculating Prospect with Program 8.3.2

The number of loops needed to calculate the convolutions with Program 8.3.2 is given by:

$$\sum_{x=1}^{C-1}\sum_{\substack{y_1=\\(x+1)\underline{f}}}^{(x+1)\overline{f}}\sum_{\substack{y_2=\\ \max(\underline{f},y_1-x\overline{f})}}^{\min(\overline{f},y_1-x\underline{f})} 1. \qquad \text{(F.4)}$$

To see how the middle (y_1) sum can be split to eliminate the min and max functions in the last (y_2) sum, note that the value of $\min(\overline{f}, y_1 - x\underline{f})$ is $y_1 - x\underline{f}$ when $y_1 \in \{(x+1)\underline{f}, \ldots, \overline{f} + x\underline{f}\}$, and $\overline{f}$ when $y_1 \in \{\overline{f} + x\underline{f}, \ldots, (x+1)\overline{f}\}$. Similarly, the value of $\max(\underline{f}, y_1 - x\overline{f})$ is $\underline{f}$ when $y_1 \in \{(x+1)\underline{f}, \ldots, \underline{f} + x\overline{f}\}$, and $y_1 - x\overline{f}$ when $y_1 \in \{\underline{f} + x\overline{f}, \ldots, (x+1)\overline{f}\}$. Figure F.1 shows this graphically. If $\overline{f} = \underline{f}$ then all the four points in Figure F.1 are the same, and that case will be handled separately.

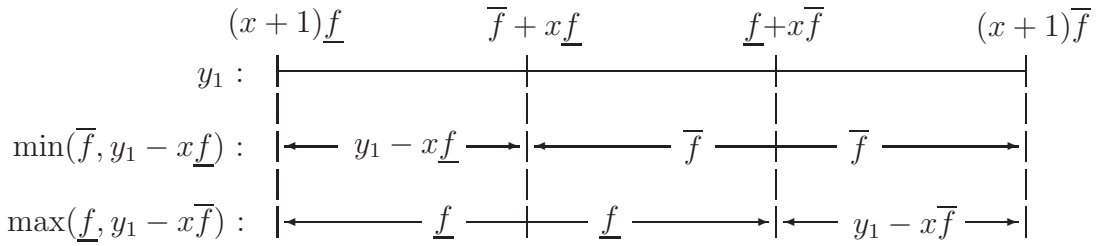


Figure F.1: Value of $\max(\underline{f}, y_1 - x\overline{f})$ and $\min(\overline{f}, y_1 - x\underline{f})$ as a function of y_1.

F.2.1 Case 1: $\overline{f} \equiv \underline{f}$

In this case, $(x+1)\underline{f} = \overline{f} + x\underline{f} = \underline{f} + x\overline{f} = (x+1)\overline{f}$ and $\min(\overline{f}, y_1 - x\underline{f}) = \max(\underline{f}, y_1 - x\overline{f}) = \overline{f}$ so the calculation is quite simple:

$$\sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ (x+1)\underline{f}}}^{(x+1)\overline{f}} \sum_{\substack{y_2= \\ \max(\underline{f}, y_1 - x\overline{f})}}^{\min(\overline{f}, y_1 - x\underline{f})} 1 = \sum_{x=1}^{C-1} 1 = C - 1. \tag{F.5}$$

F.2.2 Case 2: $\overline{f} > \underline{f}$

Here it is first shown how the middle (y_1) sum is split to eliminate the min and max functions in the last (y_2) sum, and then the three sums are evaluated separately before summing them back together again:

$$\begin{aligned} &\sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ (x+1)\underline{f}}}^{(x+1)\overline{f}} \sum_{\substack{y_2= \\ \max(\underline{f}, y_1 - x\overline{f})}}^{\min(\overline{f}, y_1 - x\underline{f})} 1 \\ &= \sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ (x+1)\underline{f}}}^{\overline{f}+ x\underline{f}-1} \sum_{\substack{y_2= \\ \underline{f}}}^{y_1 - x\underline{f}} 1 + \sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ \overline{f}+x\underline{f}}}^{\underline{f}+x\overline{f}} \sum_{\underline{f}}^{\overline{f}} 1 + \sum_{x=1}^{C-1} \sum_{\substack{y_1=\underline{f}+ \\ x\overline{f}+1}}^{(x+1)\overline{f}} \sum_{\substack{y_2= \\ y_1 - x\overline{f}}}^{\overline{f}} 1. \end{aligned} \tag{F.6}$$

Left sum:

$$\begin{aligned}
\sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ (x+1)\underline{f}}}^{\substack{\overline{f}+ \\ x\underline{f}-1}} \sum_{\substack{y_2= \\ \underline{f}}}^{y_1-x\underline{f}} 1 &= \sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ (x+1)\underline{f}}}^{\substack{\overline{f}+ \\ x\underline{f}-1}} [y_1 - (x+1)\underline{f} + 1] \\
&= \sum_{x=1}^{C-1} \sum_{\hat{y}_1=1}^{\overline{f}-\underline{f}} \hat{y}_1 \quad \text{(Change of variable, } \hat{y}_1 \equiv y_1 - (x+1)\underline{f} + 1.\text{)} \\
&= \sum_{x=1}^{C-1} \frac{(\overline{f}-\underline{f})(\overline{f}-\underline{f}+1)}{2} \\
&= (C-1)\frac{(\overline{f}-\underline{f})(\overline{f}-\underline{f}+1)}{2} \\
&= (C-1)\frac{R_f(R_f+1)}{2}. \qquad \text{(F.7)}
\end{aligned}$$

Middle sum:

$$\begin{aligned}
\sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ \overline{f}+x\underline{f}}}^{\underline{f}+x\overline{f}} \sum_{\underline{f}}^{\overline{f}} 1 &= \sum_{x=1}^{C-1} \sum_{\substack{y_1= \\ \overline{f}+x\underline{f}}}^{\underline{f}+x\overline{f}} (\overline{f}-\underline{f}+1) \\
&= (\overline{f}-\underline{f}+1) \sum_{x=1}^{C-1} (\underline{f} + x\overline{f} - \overline{f} - x\underline{f} + 1) \\
&= (\overline{f}-\underline{f}+1) \left(\sum_{x=1}^{C-1} (\underline{f} - \overline{f} + 1) + (\overline{f}-\underline{f}) \sum_{x=1}^{C-1} x \right) \\
&= (\overline{f}-\underline{f}+1) \left((\underline{f} - \overline{f} + 1)(C-1) + (\overline{f}-\underline{f}) \frac{C(C-1)}{2} \right) \\
&= (R_f+1) \left((1-R_f)(C-1) + R_f \frac{C(C-1)}{2} \right). \qquad \text{(F.8)}
\end{aligned}$$

Right sum:

$$\begin{aligned}
\sum_{x=1}^{C-1} \sum_{\substack{y_1=\underline{f}+ \\ x\overline{f}+1}}^{(x+1)\overline{f}} \sum_{\substack{y_2= \\ y_1-x\overline{f}}}^{\overline{f}} 1 &= \sum_{x=1}^{C-1} \sum_{\substack{y_1=\underline{f}+ \\ x\overline{f}+1}}^{(x+1)\overline{f}} [(x+1)\overline{f}+1-y_1] \\
&= \sum_{x=1}^{C-1} \sum_{\hat{y}_1=1}^{\overline{f}-\underline{f}} (\overline{f}-\underline{f}+1-\hat{y}_1) \quad \text{(Change of variable, } \hat{y}_1 \equiv y_1 - \underline{f} - x\overline{f}.\text{)} \\
&= \sum_{x=1}^{C-1} \left(\sum_{\hat{y}_1=1}^{\overline{f}-\underline{f}} (\overline{f}-\underline{f}+1) - \sum_{\hat{y}_1=1}^{\overline{f}-\underline{f}} \hat{y}_1 \right) \\
&= \sum_{x=1}^{C-1} \left((\overline{f}-\underline{f})(\overline{f}-\underline{f}+1) - \frac{(\overline{f}-\underline{f})(\overline{f}-\underline{f}+1)}{2} \right) \\
&= (C-1)\frac{(\overline{f}-\underline{f})(\overline{f}-\underline{f}+1)}{2} \\
&= (C-1)\frac{R_f(R_f+1)}{2}. \qquad \text{(F.9)}
\end{aligned}$$

Equations (F.7), (F.8), and (F.9) summed together and simplified equal:

$$\begin{aligned}
&2(C-1)\frac{R_f(R_f+1)}{2} + (R_f+1)\left((1-R_f)(C-1) + R_f\frac{C(C-1)}{2} \right) \\
&= (R_f+1)\left((C-1)R_f + (1-R_f)(C-1) + R_f\frac{C(C-1)}{2} \right) \\
&= (R_f+1)(C-1)\left(1 + R_f\frac{C}{2} \right). \qquad \text{(F.10)}
\end{aligned}$$

Note that Equation (F.10) reduces to Equation (F.5) from Case 1 when $R_f = 0$ ($\underline{f} = \overline{f}$), and therefore Equation (F.10) covers both Cases 1 and 2.

F.3 Updating the Prospect with Programs 8.3.3 and 8.3.4

The number of loops needed for a Tally Drop operation according to Program 8.3.3 is given by:

$$
\begin{aligned}
&\sum_{\substack{n=\\1}}^{C-1}\sum_{\substack{m=\\n+1}}^{C}\sum_{\substack{y=\\\underline{f}n}}^{\overline{f}n} 1 + \sum_{x=1}^{C} 1 \\
&= \sum_{\substack{n=\\1}}^{C-1}\sum_{\substack{m=\\n+1}}^{C} [(\overline{f}-\underline{f})n + 1] + C \\
&= (\overline{f}-\underline{f})\sum_{\substack{n=\\1}}^{C-1} n\left(\sum_{\substack{m=\\n+1}}^{C} 1\right) + \sum_{\substack{n=\\1}}^{C-1}\sum_{\substack{m=\\n+1}}^{C} 1 + C \\
&= R_f \sum_{n=1}^{C-1} n(C-n) + \sum_{n=1}^{C-1}(C-n) + C \\
&= R_f\left(C\sum_{n=1}^{C-1} n - \sum_{n=1}^{C-1} n^2\right) + \sum_{n=1}^{C-1} C - \sum_{n=1}^{C-1} n + C \\
&= R_f\left(C\sum_{n=1}^{C-1} n - \sum_{n=1}^{C-1} n^2\right) + C(C-1) - \frac{C(C-1)}{2} + C \\
&= R_f\left(\frac{C^2(C-1)}{2} - \frac{C(C-1)(2C-1)}{6}\right) + \frac{C(C+1)}{2} \\
&= R_f\frac{3C^3 - 3C^2 - 2C^3 + 3C^2 - C}{6} + \frac{C(C+1)}{2} \\
&= R_f\frac{C(C^2-1)}{6} + \frac{C(C+1)}{2} \\
&= \frac{C(C+1)}{6}[R_f(C-1) + 3].
\end{aligned}
\tag{F.11}
$$

The number of loops needed for a Tally Add operation according to Program 8.3.4 is given by:

$$
\begin{aligned}
&\sum_{m=2}^{C}\sum_{n=1}^{m-1}\sum_{y=\underline{f}n}^{\overline{f}n} 1 + \sum_{x=1}^{C} 1 \\
&= \sum_{\hat{m}=1}^{C-1}\sum_{n=1}^{\hat{m}} [(\overline{f}-\underline{f})n + 1] + C \quad \text{(Change of variable, } \hat{m} \equiv m-1.\text{)} \\
&= (\overline{f}-\underline{f})\sum_{\hat{m}=1}^{C-1}\sum_{n=1}^{\hat{m}} n + \sum_{\hat{m}=1}^{C-1}\sum_{n=1}^{\hat{m}} 1 + C \\
&= (\overline{f}-\underline{f})\sum_{\hat{m}=1}^{C-1}\frac{\hat{m}(\hat{m}+1)}{2} + \sum_{\hat{m}=1}^{C-1}\hat{m} + C \\
&= (\overline{f}-\underline{f})\left(\sum_{\hat{m}=1}^{C-1}\frac{\hat{m}^2}{2} + \sum_{\hat{m}=1}^{C-1}\frac{\hat{m}}{2}\right) + \frac{C(C-1)}{2} + C \\
&= (\overline{f}-\underline{f})\left(\frac{C(C-1)(2C-1)}{12} + \frac{C(C-1)}{4}\right) + \frac{C(C+1)}{2} \\
&= (\overline{f}-\underline{f})\frac{2C^3-3C^2+C+3C^2-3C}{12} + \frac{C(C+1)}{2} \\
&= (\overline{f}-\underline{f})\frac{C^3-C}{6} + \frac{C(C+1)}{2} \\
&= R_f\frac{C(C^2-1)}{6} + \frac{C(C+1)}{2} \\
&= \frac{C(C+1)}{6}[R_f(C-1)+3]. \qquad \text{(F.12)}
\end{aligned}
$$

The total number of loops for a Tally update (drop and add) is the sum of Equations (F.11) and (F.12):

$$\frac{C(C+1)}{3}[R_f(C-1)+3]. \qquad \text{(F.13)}$$

REFERENCES

[1] Albers, S. and Mitzenmacher, M., "Average-case analyses of first fit and random fit bin packing," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 290–299, ACM/SIAM, 1998.

[2] Asgeirsson, E. I., *On the performance of simple algorithms for solving NP-hard problems*. PhD thesis, Columbia University, New York, NY, 2007.

[3] Asgeirsson, E. I. and Stein, C., "Using Markov chains to design algorithms for bounded-space on-line bin cover," in *ALENEX*, SIAM, 2006.

[4] Asgeirsson, E. I. and Stein, C., "Bounded-space online bin cover," *J. of Scheduling*, vol. 12, no. 5, pp. 461–474, 2009.

[5] Assmann, A. B., Johnson, D. J., Kleitman, D. J., and Leung, J. Y.-T., "On a dual version of the one-dimensional bin packing problem," *Journal of Algorithms*, vol. 5, pp. 502–525, 1984.

[6] Babel, L., Chen, B., Kellerer, H., and Kotov, V., "On-line algorithms for cardinality constrained bin packing problems," in *Proceedings of the 12th International Symposium on Algorithms and Computation*, ISAAC '01, (London, UK), pp. 695–706, Springer-Verlag, 2001.

[7] Babel, L., Chen, B., Kellerer, H., and Kotov, V., "Algorithms for on-line bin-packing problems with cardinality constraints," *Discrete Applied Mathematics*, vol. 143, no. 1-3, pp. 238–251, 2004.

[8] Baker, B. S., "A new proof for the first-fit decreasing bin-packing algorithm," *Journal of Algorithms*, vol. 6, no. 1, pp. 49 – 70, 1985.

[9] Balas, E., "An additive algorithm for solving linear programs with zero-one variables," *Operations Research*, vol. 13, pp. 517–546, 1965.

[10] Balogh, J., Békési, J., and Galambos, G., "New lower bounds for certain classes of bin packing algorithms," in *Approximation and Online Algorithms* (Jansen, K. and Solis-Oba, R., eds.), vol. 6534 of *Lecture Notes in Computer Science*, pp. 25–36, Springer Berlin Heidelberg, 2011.

[11] Bender, M. A., Bradley, B., Jagannathan, G., and Pillaipakkamnatt, K., "Sum-of-squares heuristics for bin packing and memory allocation," *ACM Journal of Experimental Algorithms*, vol. 12, 2007.

[12] Bentley, J. L., Johnson, D. S., Leighton, F. T., McGeoch, C. C., and McGeoch, L. A., "Some unexpected expected behavior results for bin packing," in *Proc. 16th ACM Symposium on Theory of Computing (STOC)*, pp. 279–288, 1984.

[13] Bertsekas, D. P., *Dynamic Programming and Optimal Control*, vol. 1. Athena Scientific, 1995.

[14] Bertsekas, D. P., *Dynamic Programming and Optimal Control*, vol. 2. Athena Scientific, 1995.

[15] Bertsekas, D. P. and Castañon, D. A., "Rollout algorithms for stochastic scheduling problems," *J. Heuristics*, vol. 5, no. 1, pp. 89–108, 1999.

[16] Bertsekas, D. P., Tsitsiklis, J. N., and Wu, C., "Rollout algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 3, pp. 245–262, 1997.

[17] Bickel, D. R., "Robust estimators of the mode and skewness of continuous data," *Computational Statistics and Data Analysis*, vol. 39, pp. 153–163, 2002.

[18] Brooks, R. L., Smith, C. A. B., Stone, A. H., and Tutte, W. T., "The dissection of rectangles into squares," *Duke Mathematical Journal*, vol. 7, pp. 312–340, 1940.

[19] Cameron, P. J., *Combinatorics: Topics, Techniques, Algorithms*. Cambridge University Press, January 1995.

[20] Caprara, A. and Pferschy, U., "Worst-case analysis of the subset sum algorithm for bin packing," *Oper. Res. Lett.*, vol. 32, no. 2, pp. 159–166, 2004.

[21] Caprara, A. and Pferschy, U., "Modified subset sum heuristics for bin packing," *Inf. Process. Lett.*, vol. 96, no. 1, pp. 18–23, 2005.

[22] Coffman, Jr., E. G., Courcoubetis, C., Garey, M. R., Johnson, D. S., Shor, P. W., Weber, R. R., and Yannakakis, M., "Bin packing with discrete item sizes, part i: Perfect packing theorems and the average case behavior of optimal packings," *SIAM Journal on Discrete Mathematics*, vol. 13, no. 3, pp. 384 – 402, 2000.

[23] Coffman, Jr., E. G., Courcoubetis, C. A., Garey, M. R., Johnson, D. S., McGeoch, L. A., Shor, P. W., Weber, R. R., and Yannakakis, M., "Fundamental discrepancies between average-case analyses under discrete and continuous distributions: A bin packing case study," in *Proceedings of the Twenty Third Annual ACM Symposium OnTheory of Computing*, (New Orleans, Louisiana), pp. 230–240, ACM, 6–8 May 1991.

[24] Coffman, Jr., E. G., D. S. Johnson, P. W. S., and Weber, R. R., "Markov chains, computer proofs, and average-case analysis of best fit bin packing," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theoryof Computing*, (San Diego, California), pp. 412–421, ACM, 16–18 May 1993.

[25] Coffman, Jr., E. G., Galambos, G., Martello, S., and Vigo, D., "Bin packing approximation algorithms: Combinatorial analysis," in *Handbook of Combinatorial Optimization (Combinatorial Optimization, V. 4)* (Du, D. and Pardalos, P. M., eds.), Kluwer Academic Publishers, 1999.

[26] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S., "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, pp. 1–17, 1978.

[27] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S., "Approximation algorithms for bin–packing–an updated survey," in *Approximation Algorithms for Computer System Design* (Ausiello et al., ed.), (Wien), pp. 49–106, 1984.

[28] Coffman, Jr., E. G., Garey, M. R., and Johnson, D. S., "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems* (Hochbaum, D. S., ed.), pp. 46–93, Boston: PWS Pub. Co., 1997.

[29] Coffman, Jr., E. G. and Lueker, G. S., *Probabilistic Analysis of Packing and Partitioning Algorithms*. Wiley-Interscience Series in Discrete Mathematics and Optimization., New York: John Wiley and Sons, INC., 1991.

[30] Coffman, Jr., E. G., So, K., Hofri, M., and Yao, A. C., "A stochastic model of bin packing," *Information and Control*, vol. 44, pp. 105–115, 1980.

[31] Courcoubetis, C. and Weber, R. R., "A bin-packing system for objects with sizes from a finite set: Necessary and sufficient conditions for stability and some applications," in *Proceedings of 25th Conference on Decision and Control. Athens, Greece, Dec. 1986.*, (New York, NY, USA), pp. 1686–1691, The Institute of Electrical and Electronics Engineers, Inc., 1986.

[32] Courcoubetis, C. and Weber, R. R., "Necessary and sufficient conditions for stability of a bin-packing system," *Journal of Applied Probability*, vol. 23, pp. 989–999, 1986.

[33] Courcoubetis, C. and Weber, R. R., "Stability of on-line bin packing with random arrivals and long-run average constraints," *Probability in the Engineering and Information Sciences*, vol. 4, pp. 447–460, 1990.

[34] Csirik, J., "Bin packing as a random walk: A note on Knoedel's paper," *Oper. Res. Lett.*, vol. 5, pp. 161–163, 1986.

[35] Csirik, J., "An on-line algorithm for variable-sized bin packing," *Acta Informatica*, vol. 26, no. 8, pp. 697–709, 1989.

[36] Csirik, J., Frenk, J. B. G., Galambos, G., and Rinnooy Kan, A. H. G., "Probabilistic anaysis of algorithms for dual bin packing problems," *Journal of Algorithms*, vol. 12, no. 2, pp. 189–203, 1991.

[37] Csirik, J. and Galambos, G., "An $O(n)$ bin packing algorithm for uniformly distributed data," *Computing*, vol. 33, pp. 313–319, 1986.

[38] Csirik, J., Johnson, D. S., Kenyon, C., Shor, P. W., and Weber, R. R., "A self-organizing bin packing heuristic," in *Algorithm Engineering and Experimentation: ALENEX '99 Workshop* (McGeoch, C. and Goodrich, M., eds.), (Tiergartenstr. 17 D-69121 Heidelberg Germany), Springer-Verlag Heidelberg, 1999.

[39] Csirik, J. and Totik, V., "On line algorithms for a dual version of bin packing," *Discrete Applied Mathematics*, vol. 21, no. 2, pp. 163–167, 1988.

[40] Csirik, J. and Woeginger, G. J., "On-line packing and covering problems," in *Online Algorithms : The State of the Art* (Fiat, A. and Woeginger, G. J., eds.), ch. 7, pp. 147–177, Berlin: Springer-Verlag, 1998.

[41] Csirik, J., Johnson, D. S., and Kenyon, C., "Better approximation algorithms for bin covering," in *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA 2001)*, pp. 557–566, Society for Industrial and Applied Mathematics, 2001.

[42] Csirik, J., Johnson, D. S., and Kenyon, C., "On the worst-case performance of the sum-of-squares algorithm for bin packing," *The Computing Research Repository*, vol. abs/cs/0509031, 2005.

[43] Csirik, J., Johnson, D. S., Kenyon, C., Orlin, J. B., Shor, P. W., and Weber, R. R., "On the sum-of-squares algorithm for bin packing," in *Theory of Computing*, pp. 208–217, ACM, 2000. Proceedings of the thirty-second annual ACM symposium on Theory of computing, STOC'2000, May 21–23, 2000, Portland.

[44] Csirik, J., Johnson, D. S., Kenyon, C., Orlin, J. B., Shor, P. W., and Weber, R. R., "On the sum-of-squares algorithm for bin packing," *Journal of the Association for Computing Machinery*, vol. 53, no. 1, pp. 1–65, 2006.

[45] D., G. J. N. and C., H. J., "A new heuristic algorithm for the one-dimensional bin-packing problem," *Production Planning and Control*, vol. 10, pp. 598–603(6), 1 September 1999.

[46] Dósa, G., "The tight bound of first fit decreasing bin-packing algorithm is ffd(i) <= 11/9opt(i) + 6/9," in *ESCAPE* (Chen, B., Paterson, M., and Zhang, G., eds.), vol. 4614 of *Lecture Notes in Computer Science*, pp. 1–11, Springer, 2007.

[47] Dyckhoff, H., "A typology of cutting and packing problems," *European Journal of Operational Research*, vol. 44, pp. 145–159, 1990.

[48] Dyckhoff, H., Scheithauer, G., and Terno, J., "Cutting and packing: An annotated bibliography," in *Annotated Bibliographies in Combinatorial Optimization* (M. Dell'Amico, F. Maffioli and, S. M., ed.), pp. 393–412, New York: John Wiley and Sons, 1997.

[49] Eilon, S. and Christofides, N., "The loading problem," *Management Science*, vol. 17, no. 5, pp. 259–268, 1971.

[50] Epstein, L., "Bin packing with rejection revisited," in *WAOA* (Erlebach, T. and Kaklamanis, C., eds.), vol. 4368 of *Lecture Notes in Computer Science*, pp. 146–159, Springer, 2006.

[51] Epstein, L., Imreh, C., and Levin, A., "Bin covering with cardinality constraints," *Discrete Applied Mathematics*, vol. 161, no. 1314, pp. 1975 – 1987, 2013.

[52] Falkenauer, E., "Tapping the full power of genetic algorithm through suitable represantation and local optimisation: Application to bin packing," in *Evolutionary Algorithms in Management Applications* (Biethahn, J. and Nissen, V., eds.), pp. 167–182, New York: Springer-Verlag, 1995.

[53] Feller, W., *An Introduction to Probability Theory and Its Applications*, vol. II. New York: John Wiley & Sons, Ltd, second ed., 1971.

[54] Fleszar, K. and Charalambous, C., "Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem," *European Journal of Operational Research*, vol. 210, no. 2, pp. 176 – 184, 2011.

[55] Fleszar, K. and Hindi, K. S., "New heuristics for one-dimensional bin-packing," *Computers & OR*, vol. 29, no. 7, pp. 821–839, 2002.

[56] Floyd, S. and Karp, R. M., "FFD bin packing for item sizes with uniform distributions on $[0, \frac{1}{2}]$," *Algorithmica*, vol. 6, no. 2, pp. 222–240, 1991.

[57] Fog, A., "RANROT B Pseudo Random Number Generator." Web Page: http://agner.org/random, 1998.

[58] Foster, D. P. and Vohra, R. V., "Probabilistic analysis of a heuristics for the dual bin packing problem," *Information Processing Letters*, vol. 31, pp. 287–290, 1989.

[59] Fox, B. L. and Landi, D. M., "Scientific applications: An algorithm for identifying the ergodic subchains and transient states of a stochastic matrix," *Commun. ACM*, vol. 11, no. 9, pp. 619–621, 1968.

[60] Friesen, D. K. and Langston, M. A., "Variable sized bin packing," *SIAM Journal on Computing*, vol. 15, pp. 222–230, 1986.

[61] Fukunaga, A. S. and Korf, R. E., "Bin completion algorithms for multicontainer packing, knapsack, and covering problems," *J. Artif. Intell. Res. (JAIR)*, vol. 28, pp. 393–429, 2007.

[62] Galambos, G., "Parametric lower bound for on–line bin–packing," *SIAM Journal on Algebraic and Discrete Methods*, vol. 7, no. 3, pp. 362–367, 1986.

[63] Galambos, G. and Woeginger, G., "On-line bin packing—a restricted survey.," *ZOR—Math. Methods Oper. Res.*, vol. 42, pp. 25–45, 1995.

[64] Garey, M. R., Graham, R. L., Johnson, D. S., and Yao, A. C. C., "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory. Series A*, vol. 21, pp. 257–298, 1976.

[65] Garey, M. R., Graham, R. L., and Ullman, J. D., "Worst-case analysis of memory allocation algorithms," in *Proc. 4th Annual ACM Symp. on the Theory of Computing*, pp. 143–150, 1972.

[66] Garey, M. R., Graham, R. L., and Ullman, J. D., "An analysis of some packing algorithms," in *Combinatorial Algorithms (Algorithmic Press)* (Rustin, R., ed.), pp. 39–47, New York, 1973.

[67] Gilmore, P. C. and Gomory, R. E., "A linear programming approach to the cutting stock problem," *Operations Research*, vol. 9, pp. 849–859, 1961.

[68] Gilmore, P. C. and Gomory, R. E., "A linear programming approach to the cutting stock problem—part II," *Operations Research*, vol. 11, pp. 863–888, 1963.

[69] Gilmore, P. C. and Gomory, R. E., "The theory and computation of knapsack functions," *Operations Research*, vol. 14, pp. 1045–1075, 1966.

[70] Graham, R. L., "Bounds on multiprocessing anomalies and related packing problem," in *AFIPS Spring Joint Computer Conference*, pp. 205–217, 1972.

[71] Güntzer, M. M. and Jungnickel, D., "Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem," *Operations Research Letters*, vol. 26, pp. 55–66, March 2000.

[72] Gutin, G., Jensen, T., and Yeo, A., "Batched bin packing," *Discrete Optimization*, vol. 2, pp. 71–82, March 2005.

[73] Hjaltason, J., "Samval í öskjur," tech. rep., Faculty of Engineering, University of Iceland, 1997. Unpublished, in Icelandic.

[74] Hoffmann, U., "A class of simple stochastic on line packing algorithms," *Computing*, vol. 29, pp. 227–239, 1982.

[75] Johnson, D. S., *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1973.

[76] Johnson, D. S., "Fast algorithms for bin packing," *Journal of Computer and System Sciences*, vol. 8, pp. 272–314, 1974.

[77] Kantorovich, L. V., "Mathematical methods of organising and planning production," *Management Science*, vol. 6, pp. 366–422, 1960. (1939 Russian), (1960 English).

[78] Karmarkar, N., "Probabilistic analysis of some bin-packing problems," in *23rd Annual IEEE Symposium on Foundations of Computer Science*, (New York, NY, USA), pp. 107–111, The Institute of Electrical and Electronics Engineers, Inc., 1982.

[79] Karmarkar, N. and Karp, R. M., "An efficient approximation scheme for the one–dimensional bin–packing problem," in *23rd Annual IEEE Symposium on Foundations of Computer Science*, (New York, NY, USA), pp. 312–320, The Institute of Electrical and Electronics Engineers, Inc., 1982.

[80] Kellerer, H. and Pferschy, U., "An efficient fully polynomial approximation scheme for the subset-sum problem," *Annals of Operations Research*, vol. 92, no. 1, pp. 335–348, 1999.

[81] Kleywegt, A. J. Georgia Institute of Technology, personal communication, 2005.

[82] Knödel, W., "A bin packing algorithnm with complexity $o(nlogn)$ and performance 1 in the stochastic limit," in *Lectional Notes in Computer Science 118*, pp. 369–378, Mathematical Foundations of Computer Science, 1981.

[83] Krause, K., Larmore, L. L., and Volper, D. J., "Packing items from a triangular distribution," *Inf. Process. Lett.*, vol. 25, pp. 351–361, 1987.

[84] Krause, K. L., Shen, V. Y., and Schwetman, H. D., "Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems," *J. ACM*, vol. 22, no. 4, pp. 522–550, 1975.

[85] Krause, K. L., Shen, V. Y., and Schwetman, H. D., "Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems, errata.," *J. ACM*, vol. 24, no. 3, p. 527, 1977.

[86] Labbe, M., Laporte, G., and Martello, S., "An exact algorithm for the dual bin packing problem," *Oper. Res. Lett.*, vol. 17, no. 1, pp. 9–18, 1995.

[87] Lee, C. C. and Lee, D. T., "A simple on-line bin-packing algorithm," *Journal of the Association for Computing Machinery*, vol. 32, pp. 562–572, July 1985.

[88] Lee, C. C. and Lee, D. T., "Robust on-line bin packing algorithms," Tech. Rep. 83-03-FC-02, Department of Electircal Engineering and Computer Science, Northwestern University, Evanston, IL, 1987 (unpublished).

[89] Liang, F. M., "A lower bound for on-line bin packing," *Inf. Process. Lett*, vol. 10, pp. 76–79, 1980.

[90] Litton, C. D., "A frequency approach to the one-dimensional cutting problem for carpet rolls," *Operational Research Quarterly*, vol. 28, no. 4 ii, pp. 927–938, 1977.

[91] Loulou, R., "Probabilistic behaviour of optimal bin-packing solutions," *Operations Research Letters*, vol. 3, pp. 129–135, Aug. 1984.

[92] Lueker, G. S., "Bin packing with items uniformly distributed over intervals [a,b]," in *Proc. 24th Ann. Symp. of Comp. Sci., Tucson*, pp. 289–297, 1983.

[93] Mao, W., "Tight worst-case performance bounds for next-k-fit bin packing," *SIAM Journal on Computing*, vol. 22, pp. 46–56, February 1993.

[94] Martello, S. and Toth, P., *Knapsack Problems – Algorithms and Computer Implementations*. John Wiley & Sons, Chechester et al., 1990.

[95] Murgolo, F. D., "An efficient approximation scheme for variable-sized bin packing," *SIAM Journal on Computing*, vol. 16, pp. 149–161, 1987.

[96] Murgolo, F. D., "Anomalous behavior in bin packing algorithms," *Discrete Applied Mathematics*, vol. 21, pp. 229–243, 1988.

[97] Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc, 1994.

[98] Ramanan, P., "Average-case analysis of the smart next fit algorithm," *Inf. Process. Lett.*, vol. 31, no. 5, pp. 221–225, 1989.

[99] Ramanan, P., Brown, D. J., Lee, C. C., and Lee, D. T., "On-line bin packing in linear time," *Journal of Algorithms*, vol. 10, no. 3, pp. 305–326, 1989.

[100] Rhee, W. T. and Talagrand, M., "Some distributions that allow perfect packing," *Journal of the Association for Computing Machinery*, vol. 35, no. 3, pp. 564–578, 1988.

[101] Rhee, W. T. and Talagrand, M., "Dual bin packing with items of random sizes," *Mathematical Programming*, vol. 58, pp. 229–242, 1993.

[102] Richey, M. B., "Improved bounds for harmonic-based bin packing algorithms," *Discrete Appl. Math.*, vol. 34, no. 1-3, pp. 203–227, 1991.

[103] Ross, S. M., *Stochastic Processes*. Wiley Series in Probability and Mathematical Statistics., Wiley, New York, 1983.

[104] Runarsson, T. P., Jensson, P., and Jonsson, M. T., "Dynamic dual bin packing using fuzzy objectives," in *Proceedings of 1996 IEEE International Conference on Evolutionary Computation (ICEC '96)*, (Piscataway, N.J. USA), pp. 219–222, IEEE, Institute of Electrical and Electronics Engineers, 1996.

[105] Scholl, A., Klein, R., and Jürgens, C., "BISON: A fast hybrid procedure for exactly solving the one–dimensional bin packing problem," *Computers and Operations Research*, vol. 24, no. 7, pp. 627–645, 1997.

[106] Schwerin, P. and Wäscher, G., "The bin–packing problem: A problem generator and some numerical experiments with FFD packing and MTP," *International Transactions in Operational Research*, vol. 4, no. 5/6, pp. 377–389, 1997.

[107] Seiden, S. S., "On the online bin packing problem," *Journal of the ACM*, vol. 49, no. 5, pp. 640–671, 2002.

[108] Shor, P. W., "The average-case analysis of some on–line algorithms for bin packing," in *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 193–200, 1984.

[109] Shor, P. W., "The average–case analysis of some on–line algorithms for bin packing," *Combinatorica*, vol. 6, no. 2, pp. 179–200, 1986.

[110] Sigurdsson, S. University of Iceland, personal communication, March 2011.

[111] Sim, K. and Hart, E., "Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model.," in *Proceedgs of Genetic and Evolutionary Computation Conference 2013* (Alba, E., ed.), ACM SIGEVO, 2013.

[112] Simchi-Levi, D., "New worst case results for the bin–packing problem," *Naval Research Logistics*, vol. 41, pp. 579–585, 1994.

[113] Sweeney, P. E. and Paternoster, E. R., "Cutting and packing problems: A categorized, application–orientated research bibliography," *Journal of the Operational Research Society*, vol. 43, no. 7, pp. 691–706, 1992.

[114] Tesauro, G. and Galperin, G. R., "On-line policy improvement using Monte-Carlo search," in *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference* (Mozer, M. C., Jordan, M. I., and Petsche, T., eds.), pp. 1068–1074, The MIT Press, 1997.

[115] The R Foundation for Statistical Computing, "The R Project for Statistical Computing." http://www.r-project.org.

[116] van Vliet, A., "An improved lower bound for on–line bin packing algorithms," *Information Processing Letters*, vol. 43, pp. 277–284, 1992.

[117] Vanderbeck, F., "Computational study of a column generation algorithm for bin packing and cutting stock problems," *Mathematical Programming*, vol. 86, pp. 565–594, 1999.

[118] Wäscher, G., Haussner, H., and Schumann, H., "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109 – 1130, 2007.

[119] Xia, B. and Tan, Z., "Tighter bounds of the first fit algorithm for the bin-packing problem," *Discrete Applied Mathematics*, vol. 158, no. 15, pp. 1668 – 1675, 2010.

[120] Yue, M., "A simple proof of the inequality FFD(L) $\leq 11/9$ OPT(L)+1, $\forall$ L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica*, vol. 7, no. 4, pp. 321–331, 1991.